BIOS-моддинг

крис касперски ака мыщъх, FreeBSD@smtp.ru

если процессор это сердце компьютера, то BIOS — его душа. Прошивка определяет все! Возможности системы можно значительно расширить, если внедрить в BIOS свой собственный модуль, делающий что-то полезное, например, защищающий компьютер от вторжения, воющий с антивирусами или разблокирующий заблокированные возможности и разгоняющий машину до сверхсветовых скоростей. Одним словом, хачить BIOS не только можно, но и нужно!

alt:
если процессор это сердце компьютера, то BIOS — его душа. Модификация прошивки — это высший пилотаж хакерства, требующий знания железа и умения держать дизассемблер в руках. это дремучий лес, в котором очень легко заблудится, но я покажу кратчайший путь.

введение
или что нам понадобиться

Для экспериментов нам потребуется материнская плата со FLASH-BIOS'ом на борту. BIOS может быть любым. Главным образом мы будет говорить об Award'e, как самом "правильном" и популярном BIOS'e, однако, владельцы всех остальных не останутся в стороне. Для них мы приготовили универсальный способ внедрения, работающий со всеми BIOS'ами!

Опознать микросхему BIOS'a очень легко — на ней обычно накалена голографическая этикетка, которую необходимо оторвать, чтобы обнажить маркировку. Маркировка представляет длинный ряд цифр наподобие "28F1000PPC-12C4". Идем на http://www.datasheetarchive.com, заполняем строку запроса и получаем pdf-файл с подробным описанием чипа (так называемый datasheet). Теперь необходимо найти идентичный или совместимый чип FLASH-памяти, над которым мы, собственно, и будем экспериментировать. Его можно купить на радио-рынке или вытащить с поломанной матери.

Для "горячей" замены BIOS'a (т. е. выдергивания микросхемы с работающей платы), русские обвязывают микросхему нитками (можно, конечно, подковырнуть и отверткой, но при этом легко что-то закоротить), а вот гады-иностранцы после эпидемии "чиха" придумали специальные приспособления — chip extractor (съемщик чипов) и BIOS saviour (BIOS-спаситель). Приобрести их можно в продвинутых радиомагазинах или заказать по Интернету.

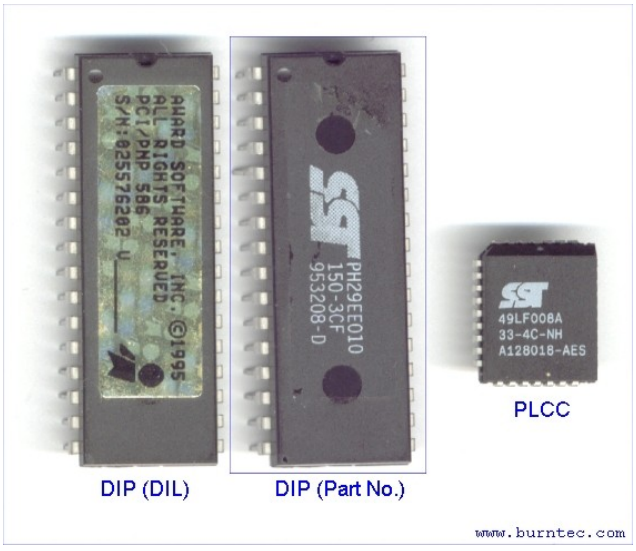

Рисунок 1 различные типы микросхем FLASH-памяти

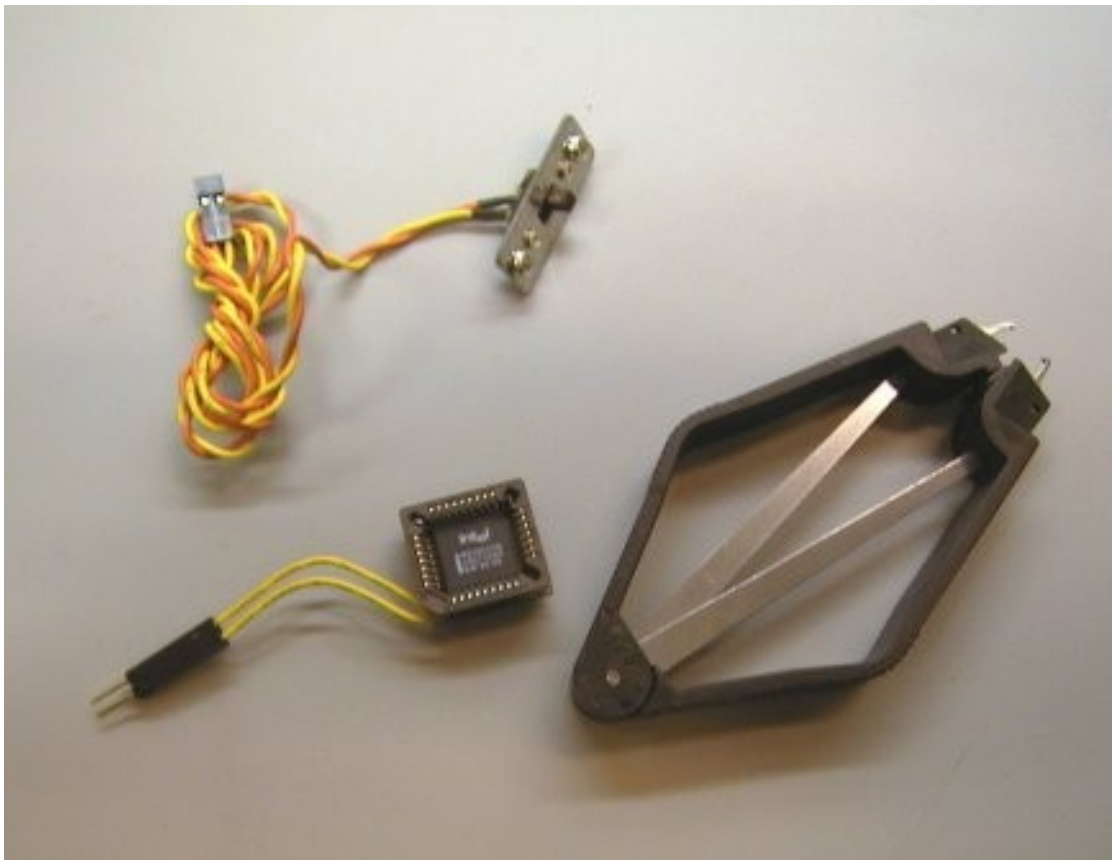

Рисунок 2 BIOS Saviour, облегчающий выемку чипа с работающей матери

Рисунок 3 BIOS extractor в действии!

Еще нам потребуется документация на чипсет материнской платы. Компании Intel и AMD бесплатно выкладывают все даташиты на сайт. Другие производители (VIA, SiS) держат их под спудом, поэтому приходится изрядно попыхтеть, прежде чем удается что-то нарыть.

>>> боковая выноска

Утилиты для работы с BIOS'ом можно найти на сайте www.rom.by. Там же находится замечательный падчер BIOS'а - BP.exe (сокращение от "BIOS Pather"), исправляющий ошибки в известных ему прошивках и разблокирующий многие заблокированные возможности.

>>> боковая выноска

Фирма Award была выкуплена Phoenix'ом и в настоящее время существует только как бренд (в смысле - торговая марка). А это значит, что Phoenix-BIOS'ы устроены точно так же как и Award, поскольку их пишет одна и та же фирма.

как мы будем действовать

Модификация BIOS'а — очень рискованное занятие (только для сильных духом мужчин!). Малейшая ошибка — и система отказывается загружаться, выдавая унылый, экран черный, как южное небо. Большинство современных матерей снабжены защитами от неудачных прошивок, однако, они срабатывают лишь тогда, когда BIOS действительно поврежден, а против ошибок в коде прошивки они бессильны.

Вот для этих целей нам и требуется второй BIOS! Запускаем материнскую плату, дампим прошивку (или скачиваем обновленную версию с сайта производителя), модифицируем ее по своему вкусу, затем, не выключая компьютера (!), аккуратно вынимаем оригинальный чип, откладывая его в сторону, вставляем чип, над которым мы будем экспериментировать, и, запустив прошивальщик, заливаем хакнутую прошивку в BIOS. Теперь, случись вдруг чего, мы всегда сможем вернуть оригинальный чип на место, исправить ошибку и перешить экспериментальный BIOS вновь.

Насколько такая процедура безопасна? По правде говоря, опасности нас подстерегают на каждом шагу. Микросхема может выскользнуть из рук и упасть на плату, малейшая ошибка в прошивке может вывести оборудование из строя (например, нечаянно "задрать" напряжение или тактовую частоту). До приобретения боевого опыта лучше всего насиловать старые материнские платы, которые все равно идут в утиль (например, Pentium-155).

>>> врезка как прожигают BIOS'ы

AMI BIOS'ы поддерживают специальный интерфейс, позволяющий работать с микросхемой FLASH-памяти (читать или прожигать), доступный через прерывания INT 15h и INT 16h (подробности — в Interrupt List'е Ральфа Брауна). Award BIOS'ы такой возможности не имеют и программируются через порты ввода/вывода. Конструктивно FLASH-микросхема подключена к южному мосту чипсета. Со всеми вопросами обращайтесь к нему, а точнее к его документации. Существует множество готовых прошивальщиков, поддерживающих практически все известные типы BIOS'ов, причем, некоторые из них распространяются вместе с исходными текстами. Одним из таких прошивальщиков является знаменитый UNIFLASH (http://www.uniflash.org/).

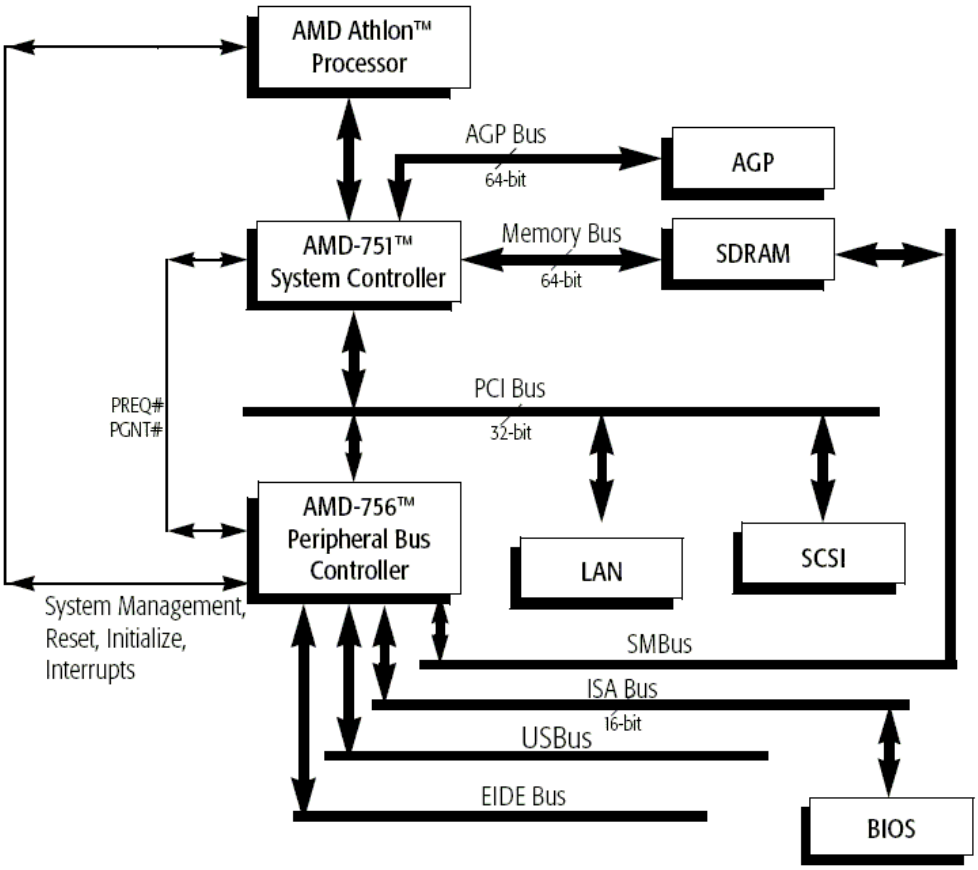

Рисунок 4 микросхема FLASH-памяти, подключенная к южному мосту

>>> врезка DUAL-BIOS своим руками

Всякий, умеющий держать паяльник в руках, может доработать материнскую плату, установив на нее сразу две микросхемы FLASH–памяти. Тогда, между ними можно будет переключаться без рискованных манипуляций с chip-extractor'ом. Пример схемы подключения приведен ниже. Как видно, ничего сложного в DUAL-BIOS'е нет.

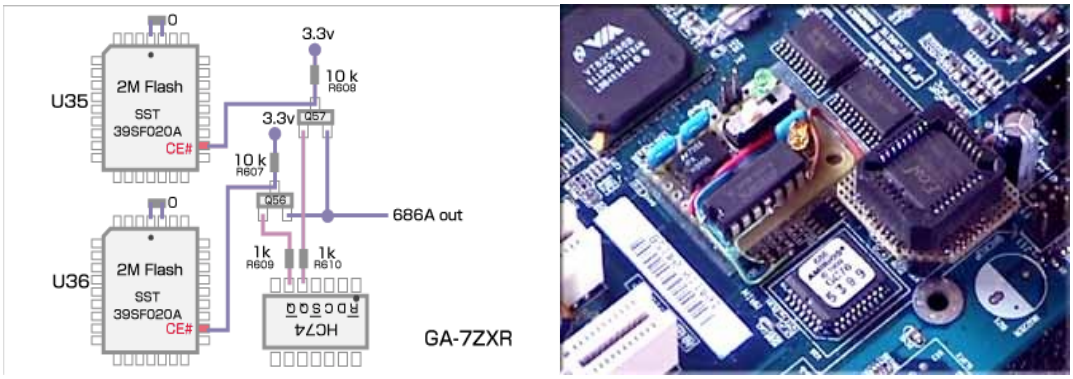

Рисунок 5 принципиальная схема DUAL-BIOS'а и ее материальное воплощение

BIOS изнутри

Чтобы модифицировать BIOS, необходимо знать его структуру. Настоящие хакеры модифицируют непосредственно машинный код, дописывая все необходимые компоненты вручную. В древних AT весь BIOS умещался в последнем сегменте адресного пространства, простилающимся от F000:0000 до F000:FFFF. Современные прошивки занимают порядка 256 —

512 Кб и чтобы обеспечить обратную совместимость, BIOS пришлось разбить на несколько частей, в результате чего он приобрел сложную модульную структуру, с которой не так-то просто разобраться.

Но мы же не боимся трудностей, верно? Вот перед нами лежит прошивка (пусть для определенности это будет прошивка 06/19/2003-i845PE-W83627-9A69VPA1C-00, с именем файла 4PE83619.BIN). Как дизассемблировать ее?! IDA едет крышей и ничего путного не показывает. Будем исходить из того, что последний байт прошивки расположен по адресу F000h:FFFFh, а точка входа в BIOS находится по адресу F000h:FFF0h.

Загружаем файл в HIEW или ИДУ, отсчитываем 10h байт от его конца и дизассемблируем код. С вероятностью близкой к единице, там будет торчать межсегментный переход:

```
0007FFD0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0007FFE0:  00 00 00 00-00 00 00 00-39 41 36 39-56 50 41 31
0007FFF0:  EA 5B E0 00-F0 2A 4D 52-42 2A 02 00-00 00 60 FF
```

Рисунок 6 HEX-дамп последних 30h байт прошивки (машинная команда, расположенная в точке входа выделена красным цветом)

```
0007FFF0: EA5BE000F0          jmp     0F000:0E05B
0007FFF5: 2A4D52              sub     cl,[di][00052]
```

Рисунок 7 дизассемблерный листинг окрестностей точки входа в прошивку

В данном случае переход указывает на адрес F000h:E05Bh. Как найти это место в прошивке? Да проще простого, если 7FFF0h это F000h:FFF0h, то 0F000h:0E05Bh это: 7FFF0h - (FFF0h - E05Bh) ==7FFF0h - 1F95h = 7E05Bh. Здесь расположен следующий код:

```
0007E05B: EA60E000F0          jmp     0F000:0E060
0007E060: 8EEA                mov     gs,dx
0007E062: FA                  cli
0007E063: FC                  cld
0007E064: 8CC8                mov     ax,cs
0007E066: 8ED0                mov     ss,ax
```

Рисунок 8 начало дизассемблирования boot-блока

Место, в которое мы попали, называется загрузочным блоком (boot-block или boot-kernel). Загрузочный блок, предельно допустимый размер которого составляет 64 Кб, выполняет первичную инициализацию оборудования и загружает все остальные блоки, хранящиеся в упакованном виде.

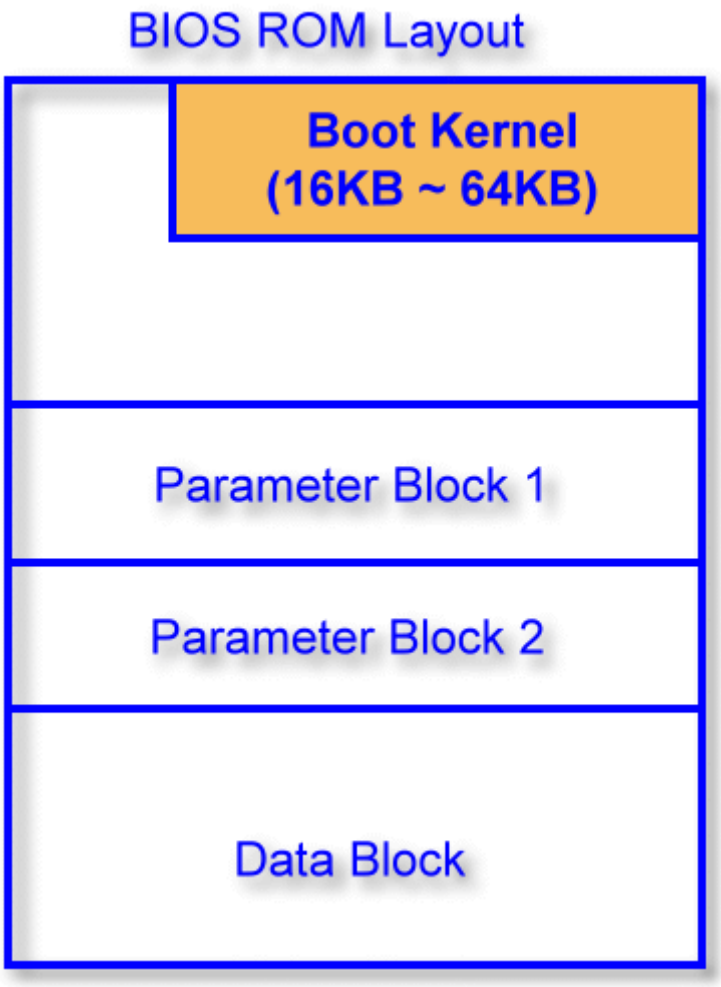

Рисунок 9 вот так или примерно так устроен Award BIOS

Естественно, перед дизассемблированием остальных блоков их необходимо распаковать. Нормальный распаковщик можно написать и самостоятельно, однако, проще взять уже готовый и автоматически извлечь все модули. В частности, обладатели Award'a могут запустить cbrom с ключом /D (или bp.exe с ключом /e), чтобы узнать из каких блоков состоит данная версия BIOS и по каким адресам они располагаются. (Утилиту cbrom можно найти на диске, прилагаемом к материнской плате или скачать с сайта www.rom.by).

В нашем случае картина будет выглядеть так:

```
C:\WINNT\system32\CMD.EXE - cmd /e auto a; auto b; b=0;
Attention! Advanced qualification is required!

Found 4Mbit BIOS!

================================================================================
BIOS-PartName Segm:Offs  Compress/Real_Size "Official" name - what's meaning
--------------------------------------------------------------------------------
9a69vpa1.BIN  5000:0000 0001:471E/0002:0000 "original.tmp" - MAIN part
awardext.rom  407F:0000 0000:81F2/0000:BA80 "awardext.rom" - ALT part
CPUCODE.BIN   4001:0000 0000:376A/0000:3800 "cpucode.bin" - microcodes
ACPITBL.BIN   4003:0000 0000:196E/0000:4068 "ACPITBL.bin" - ACPI table
awardeyt.rom  400E:0000 0000:3D7A/0000:5ED0 "awardeyt.rom" - ALT_2 part
_EN_CODE.BIN  4029:0000 0000:1A16/0000:3890 "_en_code.bin" - engl-txt Setup
AWDFLASH.EXE  4026:0000 0000:5302/0000:990C
pxe.lom       4086:0000 0000:5EA3/0000:A000 "PCI.rom" ~ SCSI-BIOS
--------------------------------------------------------------------------------
CPU_microcodes (CPUID/version/type):
--------------------------------------------------------------------------------
0F12>002C/Sock  0F21>0001/Sock  0F23>0008/Sock  0F24>0018/Sock  0F13>0004/Sock
0F27>0033/Sock  0F29>0011/Sock
================================================================================

if you can't see all messages - choose 80x50 mode or run with ">report.txt".
(c)2002-2003 apple_rom, www.ROM.by
D:\hacker\BIOS\bin\romby>_
```

Рисунок 10 результат работы утилиты BP.EXE, распечатывающий список модулей, слагающих BIOS, с их основными характеристиками

Ух ты, сколько всего! Первым идет 9a69vpa1.BIN (original.tmp). Основной код BIOS'a сосредоточен именно здесь. Как мы видим, original.tmp загружается по адресу 5000h:0000h и занимает 128Кбайт. Да-да, BIOS загружается в оперативную память, впрочем, надолго он в ней не задерживается и перед передачей управления загрузочному сектору (на винчестерах это MBR, на дискетах — boot) он обязан освободить эти адреса.

Следом идет awardext.rom (ALT part). Это расширение основного BIOS, которое инициализирует оборудование на финальной стадии загрузки (автодетектирует жесткие диски и оптические приводы, выводит таблицу PnP/PCI устройств и т. д.), а в модуле awardeyt.rom (ALT_2 part) содержится его продолжение.

CPUCODE.BIN — это просто набор микрокодов для всех поддерживаемых BIOS'ом моделей процессоров. Микрокоды предназначены для исправления ошибок, допущенных при их разработке, однако, для нормальной работы системы этот модуль в общем-то не критичен. Хотя при желании можно скачать свежую версию микрокодов с сайта Intel или AMD и залить их в BIOS.

ACPITBL.BIN — еще один модуль данных. Здесь содержится таблица ACPI (Advanced Configuration and Power Interface – Улучшенный Интерфейс Питания и Конфигуации). ACPI это отнюдь не простой менеджер питания (как наивно полагают многие). Это еще и корневой перечислитель, грубо говоря, самая главная шина, управляющая всеми устройствами и автоматически распределяющая системные ресурсы (в частности — прерывания). Модификация этой таблицы открывает большие возможности, однако, это тема для отдельного разговора.

_EN_CODE.BIN – набор текстовых ASCIIZ-строк, используемых BIOS'ом. Вот где можно развернуться начинающим хакерам!

AWDFLASH.EXE — утилита для прошивки BIOS. Вполне окейный исполняемый файл, запускаемый из-под MS-DOS. Так что для написания универсального прошивальщика BIOS ничего кроме этого самого BIOS [на хрен] не нужно! Для написания вирусов, поражающих BIOS, это актуально.

pxe.lom — хотя BP.EXE опознала этот модуль как "SCSI", текстовые строки внутри него показывают, что в действительности это набор драйверов для интегрированных устройств, а именно: VIA VT6105 Rhine III Fast Ethernet Adapter, VIA VT6105M Rhine III Management Adapter, Intel UNDI, PXE-2.0 (build 082).

Помимо вышеперечисленных, в BIOS могут входит и другие модули, например: VGA BIOS для поддержки интегрированного видео, anti_vir.bin для защиты загрузочных секторов от вирусов, decomp_blk.bin — обособленный LHA-распаковщик и т.д. Мы так же можем добавлять свои собственные модули для поддержки ISA и PCI-устройств, чем мы в последствии с успехом и воспользуемся (естественно, никаких "своих" устройств у нас нет, это просто один из многих способов внедрения постороннего кода в BIOS).

Впрочем, модификаций BIOS'a мы займемся потом, после того как разберемся с уже имеющимся модулями. Чтобы начать дизассемблирование, необходимо найти точку входа. А как ее найти? Существует по меньшей мере пять основных типов модулей. У первых точка входа расположена по смещению 10h байт от конца модуля. Если здесь находится jmp, значит, это наш клиент! (Так, в частности, устроены загрузочный и основные блоки).

```
00000000:  42 73 47 05 00 E0 00 F0 | 00 10 00 40 00 40 00 00
00000010:  1E B8 40 00 8E D8 C6 86 | 4F 02 00 F7 06 10 00 01
00000020:  00 0F 84 ED 00 B0 02 BA | F7 03 EE EB 00 EB 00 FA
…
0001FFD0:  88 1E 00 01 1F 61 CF 00 | 00 50 43 49 2F 49 53 41
0001FFE0:  00 60 03 3C E7 45 84 01 | 00 01 80 00 80 05 3E 93
0001FFF0:  EA 5B E0 00 F0 30 36 2F | 31 39 2F 30 33 00 FC E5
```

Рисунок 11 модуль 9a69vpa1.BIN с точкой входа по смещению 10h от конца (первый байт точки входа выделен красным)

Другие блоки содержат сигнатуру 55h AAh в своем начале. Если это так, следующий байт определяет длину модуля, выраженную в 512-байтовых секторах (например, 10h секторов соответствует 8 Кбайтам). Точка входа находится по смещению 03h от начала и обычно здесь стоит jmp. Так, в частности, устроены pxe.lom, i815.vga, все ISA/PCI-модули и многие другие блоки.

```
00000000:  55 AA 50 E8 1E 16 CB 8F | F9 03 00 00 00 00 00 00
```

```
00000010:  00 00 00 00 00 00 20 00 | 40 00 60 00 2E 8B C0 90
00000020:  55 4E 44 49 16 39 00 00 | 01 02 BD 10 00 08 60 97
```

Рисунок 12 модуль pxe.lom с сигнатурой 55 AA

Блоки третьего типа начинаются с обыкновенного текстового (или нетекстового) заголовка, заканчивающегося нулем. За нулем находится точка входа, опять-таки в подавляющем большинстве случаев представляющая старый добрый jmp. По такому принципу устроены awardext.rom, decomp_blk.bin, anti_vir.bin…

```
00000000:  3D 20 41 77 61 72 64 20 | 44 65 63 6F 6D 70 72 65   = Award Decompre
00000010:  73 73 69 6F 6E 20 42 69 | 6F 73 20 3D 00 66 60 51   ssion Bios = ...
00000020:  06 56 A1 04 01 80 E4 F0 | 80 FC F0 75 3A E8 B2 0A   xxxxxxxxxxxxxxxx
```

Рисунок 13 модуль decomp_blk.bin с текстовым заголовком, завершаемым нулем в начале

Блоки четвертого типа лишены заголовка и сразу же начинаются с точки входа. Наглядный пример тому — awardeyt.rom.

```
00000000:  E9 00 00 90 EA 09 00 00 | A8 8C C8 8E E0 B8 00 A0
00000010:  8E D0 66 BC F0 EF 00 00 | B8 00 F0 8E C0 BE B7 D2
00000020:  26 0F 01 1C 66 60 1E 06 | 0F A0 0F A8 BA F8 0C 66
```

Рисунок 14 модуль awardeyt.rom с точкой входа в самом начале

Блоки пятого типа вообще не имеют точки входа и представляют собой набор вспомогательных процедур, вызываемый из остальных блоков, поэтому их следует дизассемблировать в последнюю очередь, когда структура остальных блоков уже ясна.

Поскольку в BIOS'е интенсивно используются абсолютные адреса, каждый дизассемблируемый блок должен быть загружен по своему "родному" смещению, иначе у нас ничего не получится. IDA автоматически запрашивает сегмент:смещение двоичных файлов перед загрузкой, а вот в HIEW'е для этой же цели можно использовать базирование. Ну не считать же все адреса каждый раз вручную, верно? Мы же хакеры, а не лошади!

Сложнее всего дизассемблирование INTEL BIOS'ы. Фактически, это AMI BIOS'ы, но какие-же они извращенные. Точка входа лежит в середине файла и чтобы ее найти необходимо отыскать последовательность FA/FC/8C C8/8E D0 (CL/CLD/MOV AX,CS/MOV SS,AX). Собственно, это не совсем точка входа, но нечто очень к ней приближенное. Впрочем, для наших целей она вполне подходит.

боевое крещение или ISA-ROM модуль

Разобравшись с устройством BIOS'а, мы можем написать свое собственное расширение. Проще всего добавить к BIOS'у нестандартный ISA ROM модуль. Обычно такие модули используются для управления интегрированными ISA-контроллерами (например, дополнительным COM-портом). Разумеется, никакого ISA-контроллера у нас нет. ISA-слоты давно исчезли с материнских плат, но ISA-модули по-прежнему поддерживаются BIOS'ом (программистам лень править отлаженный код). ISA-модуль загружается после того, как отработает основной код BIOS (original.tmp), и получает полный доступ ко всему оборудованию, в том числе и PCI-шине. (При желании можно добавить и PCI-модуль, однако это намного сложнее. Потребуется взвести регистр XROMBAR — Expansion ROM Base address — и подделать идентификатор PCI-устройства в заголовке модуля так, чтобы он совпадал с идентификатором реально существующего устройства).

ISA модуль представляет собой обычный двоичный файл с размером кратным 200h байт, всегда загружающийся по адресу xxxx:0000h. Часть оборудования (оперативная память, клавиатура, видеокарта) к моменту его вызова уже инициализирована, а часть (жесткие диски, например) еще нет. Прерывания INT 10h (видео) и INT 16h (клава) можно использовать не боясь, а вот с INT 13h (диски) ничего не получится (то есть, получится, конечно, но не сразу и не просто так).

В начале ISA-модуля расположен стандартный 55 AA-заголовок, о котором мы уже говорили, а в последнем байте хранится контрольная сумма. Простейший ISA-ROM модуль, написанный на FASM'е, выглядит так:

```
; ISAOEM.ASM
use16                   ; ISA-модуль работает в 16-разрядном сегменте
```

```
DB      55h, 0Ah        ; загрузочная сигнатура
DB      01h             ; размер блока в 200h байтовых секторах
jmp     x_code          ; передача управления нашему коду

x_code:
        ; подготовка регистров
        ; --------------------
        MOV DX, 101Dh   ; куда выводить (DH - Y, DL - X)
        MOV SI, text    ; что выводить
        XOR BX,BX       ; начальный цвет символов — 1
        MOV CX,1        ; выводим по одному символу

        ; вывод строки в цвете
        ; --------------------
print_string:
        MOV AH, 02h     ; функция управления курсором
        INT 10h         ; позиционируем курсор
        INC DL          ; перемещаемся на следующую позицию

        LODSB           ; загружаем очередной символ
        TEST AL,AL      ; конец строки?
        JZ input        ; если конец, то выходим

        MOV AH, 09h     ; функция печати символа
        INC BL          ; перебираем все цвета
        INT 10h         ; печатаем символ
        JMP print_string;мотаем цикл

input:  ; ожидание ввода пароля
        ; ----------------------
        XOR DX,DX       ; контрольная сумма
enters:
        XOR AX,AX       ; функция чтения символа с клавы
        INT 16h         ; читаем символ
        CMP AL,0Dh      ; это ENTER?
        JZ input        ; если ENTER начинаем ввод сначала
        XOR AH,AH       ; очистить скэн-код
        ADD DX,AX       ; считаем CRC
        CMP DX, 'm' + 's' + 'o' + ']' + '['
        JNZ enters      ; если это не "мыщъх", продолжаем ввод
        retf
text db "Matrix has you!",0
```

Листинг 1 ISA-модуль. При загрузке компьютера выводит приветствие на экран и ждет слова "мыщъх", набранного на латинской раскладке (ENTER — начать ввод заново). Что-то вроде дополнительной парольной защиты, которую без выдирания BIOS'а никто не взломает!

Попустив исходный файл через транслятор ("FASM ISAOEM.ASM"), мы получим на выходе ISAOEM.BIN. Загружаем его в HIEW и дополняет нулями до размера кратного 200h байт, затем рассчитываем контрольную сумму. Контрольная сумма рассчитывается стандартным образом: мы просто складываем все байты друг с другом и находим остаток от деления на 100h. Т. е sum = (sum + next_byte) & 0xFF. Контрольная сумма всего блока должна равняться нулю, следовательно, последний байт блока равен (100h – sum) & 0xFF. Для расчета контрольной суммы я написал нехитрый скрипт для IDA:

```
auto a; auto b; b=0;
PatchByte(MaxEA()-1, 0);
for(a=MinEA();a<MaxEA();a++)
{
        b = (b + Byte(a)) & 0xFF;
}
b = (0x100 - b) & 0xFF ; Message("\n%x\n",b);
PatchByte(MaxEA()-1, b);
```

Листинг 2 IDA-скрипт, автоматически рассчитывающий контрольную сумму

Как вариант, можно использовать Hew Workshop (tools → Generate Check sum → 8 bit checksum). В нашем случае Hew Workshop сообщает CFh, следовательно, последний байт равен: 100h – CFh == 31h. Записываем его по смещению 1FFh и сваливаем из HIEW'a. Добавляем новый модуль в BIOS (CBROM.EXE 4PE83619.BIN /ISA ISAOEM.bin) и с замиранием сердца прожигаем BIOS утилитой UNIFLASH или любой другой. Перезагружаем машину…

Если все было сделано правильно, экран должен выглядеть приблизительно так:

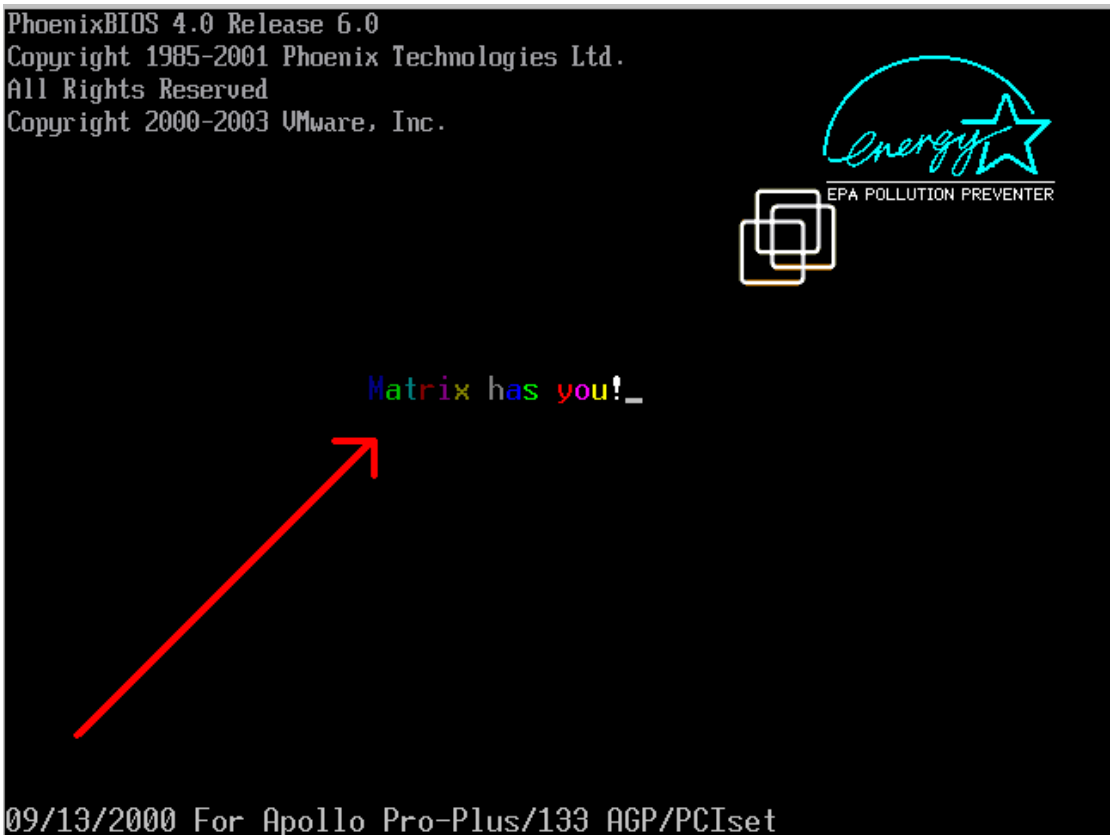

Рисунок 15 хакнутый BIOS, ожидающий ввода секретного пароля

Оторвать мыщъх'у хвост! Это работает!!! (а если не работает, то оторвать и подавно). Теперь поговорим о том, как захачить жесткие диски (например, написать boot-вирус, которой бы автоматически восстанавливался после форматирования). Прерывание INT 13h здесь не поможет, поскольку ISA-блок отрабатывает до инициализации дисков, и приходится писать резидент (а еще говорят, что вирусы в BIOS'e не живут! еще как живут! даже резидентством занимаются). Основной код BIOS'a всегда загружает boot/MBR сектор по адресу 0000:7C000h и передает ему управление. Установив на этот адрес аппаратную точку останова, мы "всплывем" в тот момент, когда все оборудование уже инициализировано и работает как часы:

Весь вопрос в том, куда спрятать наш код. По умолчанию ISA-блок распаковывается в оперативную память, которая впоследствии злобно затирается всеми кому не лень. Давным-давно, когда землей владели динозавры, а на компьютерах стояла MS-DOS, многие вирусы размешались внутри таблицы прерываний, верхняя половина которой остается незадействованной и по сей день. От адреса 0000:01E0h до ~0000:384h простилается ничейная область, в которой можно разместить ~360 байт своего обработчика. Для наших целей этого вполне достаточно.

Следующий код устанавливает аппаратную точку и перехватывает прерывание INT 01h, которое генерируется при передаче управления на загрузочный сектор. Обработчик прерывания пусть каждый пишет самостоятельно. Фактически он представляет собой обыкновенный boot-вирус, образец которого легко найти в сети.

```
MOV ax, CS                          ; перехватываем INT 01h
XOR bx,bx
MOV DS,bx
MOV [bx], offset our_vx_code        ; смещение нашего обработчика
MOV [bx+2],bx                       ; относительно сегмента 0000h
MOV DS, ax

MOV eax,302h                        ; устанавливаем точку останова на исполнение
MOV ebx,7C00h                       ; линейный физический адрес точки останова

MOV dr7,eax                         ; Заносим значения в отладочные регистры
mov dr0,ebx
```

Листинг 3 перехватчик. передает управление нашему коду в момент загрузки Boot-сектора

>>> выноска

Для программирования BIOS'а необходимо знать основные прерывания как свои пять пальцев. Вот пара отличных руководств по теме:

- Архитектура ввода-выводаперсональных ЭВМ IBM PC http://redlib.narod.ru/asmdocs/asm_doc_07.zip (описание устройства компьютера для начинающих на русском языке);
- Ralf Brown Interrupt List http://www.ctyme.com/rbrown.htm (справочник по прерываниям для профессионалов на английском языке)

BIOS на виражах или модификация boot-блока

Вышеописанный способ внедрения работает только на Award и отчасти Phoenix, что не есть хорошо. Существует универсальный способ, совместимый со всеми BIOS'ами, однако, он далеко не так прост. Единственное место, куда мы можем внедриться — это boot-блок, а точнее безусловный переход, лежащий по адресу F000h:FFF0h.

Давайте возьмем какой-нибудь BIOS и дизассемблируем его (пусть для определенности это будет AMI 6728 ver. 52 от материнской платы MSI 865PE Neo3-F, имя файла прошивки A6728IMS.520).

```
0007FD20:  80 00 00 00-00 31 49 38-36 35 78 78-78 00 00 00
0007FD30:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
…
0007FF30:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
0007FFE0:  31 49 38 36-35 58 58 58-00 00 00 00-00 00 00 00
0007FFF0:  EA CD FF 00-F0 31 31 2F-31 36 2F 30-34 00 FC 00
```

Листинг 4 внешний вид типичного boot-блока (фрагмент)

В хвосте boot-блока находится ~200h нулей, что вполне достаточно для размещения нашего кода. Исправляем EA CD FF 00-F0 (jmp 0F000:0FFCD) на EA 30 7F 00 F0 (jmp 0F000:7F30), копируем свой код поверх нулей и радуемся жизни. Естественно, в других прошивках эти цифры могут несколько отличаться, поэтому перед внедрением в BIOS наш код должен автоматически находить длинную последовательность нулей в его хвосте. Это легко. Сложнее пересчитать контрольную сумму. Различные BIOS'ы хранят ее в различных местах. Что делать? Мат. часть учить вот что! Контрольная сумма boot-блока равна нулю. Это закон. Поэтому нам достаточно рассчитать контрольную сумму нашего кода и добавить к его концу два байта (у boot-блоков, в отличие от ISA, подсчет контрольной суммы ведется не по байтам, а по словам), добившись чтобы его контрольная сумма равнялась нулю, тогда и контрольная сумма всего boot-блока будет равна нулю! Искать местоположение оригинальной контрольной суммы не нужно!

Итак, мы в boot-блоке. Ну и что мы будем делать? А ничего! Никакие устройства еще не инициализированы, даже стека нет! Оперативная память также не подготовлена, поэтому установка аппаратных точек останова ничего не даст, мы просто не сможем перехватить вектор прерывания! Это конец? Вовсе нет, это только начало! Мы знаем, что всякий boot-блок не зависимо от своей половой принадлежности выполняет первичную инициализацию оборудования, в частности, подключает внешние BIOS'ы, находящиеся на картах расширения. И делает это он тогда, когда оперативная память уже подготовлена к работе.

В нашем случае, эта конструкция находится по смещению 7078Dh от начала файла.

```
0007078D: 26813F55AA   cmp     w,es:[bx],0AA55
00070792: 7410         je      0000707A4
```

Листинг 5 "волшебная" последовательность 55 AA 7x

Все что нам надо — это найти последовательность типа "55 AA 7x ??" (CMP XXX, AA55h) и заменить "7x ??" на "EB xx" (JMP SHORT xxx, где xxx — указатель на наш код, внедренный в boot-блок). Естественно, перед затираем "7x ??" его необходимо сохранить в своем коде. Вот теперь можно устанавливать перехватчик на boot-сектор. Причем, поскольку наш код находится в BIOS'е в неупакованном виде, ютиться в таблице прерываний совершенно необязательно и можно перенаправить вектор прерывания INT 01h прямо в BIOS!

>>> врезка преодолевая барьеры

Мы можем перехватить прерывание INT 13h (и не давать его изменять!) оставаясь в BIOS'e резидентно. Только что это дает? Windows не использует INT 13h и потому наш код будет активен только на стадии первичной загрузки операционной системы. Но ведь мы же можем читать/писать сектора? Правда, писать собственный драйвер файловой системы нам лень. Хорошо, если это FAT, а как быть с NTFS? Да как два пальца об асфальт! Последовательно сканируя сектора, находим сектор с сигнатурой "MZ" в начале. Смотрим, если за концом EXE-заголовка, расположена "PE"-сигнатура, значит, это PE-файл и мы можем внедряться в него любым приемлемым способом. Внедряться лучше всего в PE-заголовок, поскольку файл может быть фрагментирован и не факт, что все последующие сектора принадлежат ему, а не какому ни будь другому ни в чем не повинному файлу.

Чтобы гарантированно получить управление, необходимо внедриться в большое количество файлов. Конечно, системные файлы будут немедленно вылечены SFC, а со всеми остальными справится антивирус, однако, и сама SFC и сам антивирус являются обыкновенными исполняемыми файлами, которые запускаются уже после того, как наш код получит управление. Естественно, он может противостоять всем зловредно настроенным программкам, например, просто блокировать их выполнение.

>>> разгон системы

Чтобы разогнать систему, необходимо написать свой ROM-модуль, настраивающий чипсет на максимальную производительность. Конфигурирование чипсета осуществляется через специальные регистры, находящиеся глубоко внутри чипсета и подключенные к шине PCI. Описание регистров можно найти в даташите. Где-то там будет раздел "PCI Configuration Registers" или что-то в этом роде. Сравнение конфигурационных возможностей чипсета с BIOS Setup показывает, что часть настроек обычно бывает умышленно заблокирована производителем материнской платы.

В частности, регистр 80000064h чипсета VIA Apollo Pro 133 управляет чередованием банков памяти (чередование существенно увеличивает производительность), в то время как многие материнские платы на его основе, такой возможности не имеют. А как ее поиметь?

У PCI-шины есть два замечательных порта. В порт CF8h заносится адрес чипсетного регистра с которым мы хотим работать, а через порт CFCh происходит обмен данными. Большинство чипсетных регистров представляют собой набор управляющих битов, поэтому перед тем как что-то записывать в порт CFCh, мы сперва должны прочитать текущее состояние чипсета, взвести/опустить нужные нам биты при помощи операций OR и AND, после чего затолкать обновленный регистр на место.

На языке ассемблера это выглядит так:

```
mov eax, 80000064h     ; регистр чипеста, управляющий DRAM-контроллером
mov dx, 0CF8h          ; PCI-порт (адрес регистра)
out dx, eax            ; выбираем регистр

mov     dx, 0cfch      ; PCI-порт (данные)
in      eax, dx        ; читаем содержимое регистра 80000064h
or      eax, 00020202h ; взводим биты, устанавливающие режим чередования
out     dx, eax        ; записываем регистр чипсета
```

Листинг 6 расширение BIOS'a, задействующее режим чередования DRAM-банков (только для чипсета VIA Apollo Pro 133! обладатели других чипсетов должны заменить подчеркнутые константны в соответствии со своей документацией)

Данный модуль может быть оформлен либо как ISA-ROM, либо внедрен в boot-блок. Главное, чтобы он получил управление после того, как BIOS произведет первичную инициализацию оборудования, иначе наши настройки будут проигнорированы! Зашив обновленную прошивку в BIOS, мы с удовлетворением замечаем, что быстродействие системы ощутимо возросло. Таким же точно образом можно редактировать и остальные регистры, отсутствующие в BIOS Setup, разгоняя систему до скорости реактивного гепарда которому в известное место залетел шмель.

Собственно говоря, это даже не разгон, а законное использование тех возможностей, которые чипсет нам дает по праву (недокументированные возможности чипсета это отдельный разговор).

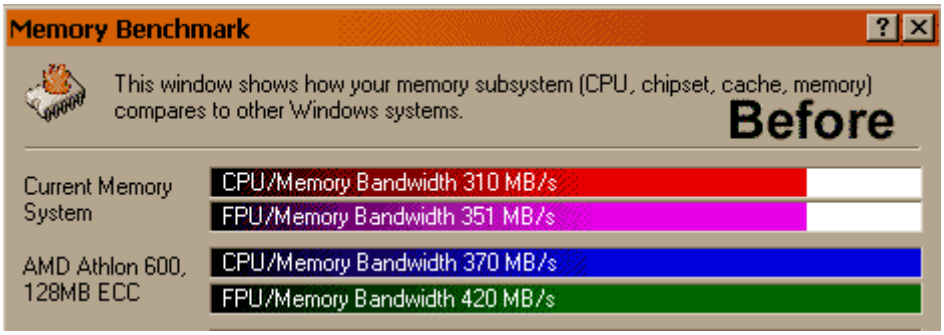

Рисунок 16 пропускная способность подсистемы памяти с режимом чередования по умолчанию

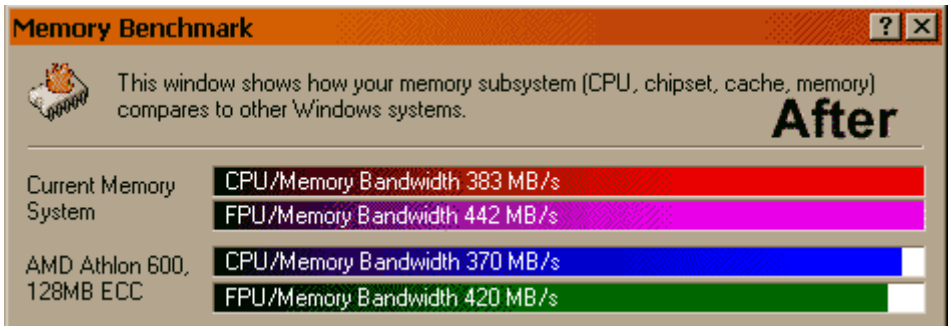

Рисунок 17 пропускная способность подсистемы памяти после разгона

заключение

Чем больше хакаешь BIOS, тем больше хочется его хакать! Это самый низкий уровень и жить на нем очень интересно и познавательно. Это настоящая школа программирования с практически неограниченными возможностями для самовыражения. Главное — фантазию иметь! Полет нашей мысли сдерживает только железо, мы можем свободно переходить в защищенный режим, крутить любые регистры и вообще делать все, что нам вздумается! И мы реально сделали это!

>>> врезка интересные ссылки

- **BIOSmods**
  - портал, посвященный BIOS'у и его доработке (на английском языке): http://www.biosmods.com/;
- **ROM**
  - статьи по прошивке и доработке прошивок, уникальный инструментарий (на русском языке): http://www.rom.by;
- **Часто задаваемые вопросы о BIOS**
  - довольно устаревшее, но все-таки полезное FAQ по BIOS'ам (на русском языке): http://www.ixbt.com/mainboard/faq/biosfaq.shtml;
- **The Official Website of Pinczakko**
  - сайт улетного индонезийского хакера, исследовавшего кучу BIOS'ов вдоль и поперек и вытворяющий с ними такое, что другим даже не снилось (на английском и индонезийском языках): http://www.geocities.com/mamanzip/;
- **Modification of GigaByte GA-586HX BIOS rev 2.9 for support of HDD above 32 GiB**
  - интересная статья о модификации BIOS, название которой говорит само за себя (на английском языке): http://www.ryston.cz/petr/bios/ga586hx_mod.html;
- **Award BIOS Reverse Engineering**
  - статья известнейшего хакера BIOS'a Mappatutu Salihun Darmawan по внедрению своего кода в BIOS (на английском языке):http://www.codebreakers-journal.com/include/getdoc.php?id=83&article=38&mode=pdf;
- **Award BIOS Code Injection**

- новые идеи по внедрению своего кода в BIOS (на английском языке): http://www.codebreakers-journal.com/include/getdoc.php?id=127&article=58&mode=pdf;

- **AWARD BIOS Source**
  - исходные тексты пары устаревших прошивок со скупыми комментариями. помогают понять общие принципы функционирования BIOS'а и разобраться в некоторых тонких местах, неочевидных при дизассемблировании: (на языке ассемблера): http://miscellaneous.newmail.ru/;
- **Редактируем BIOS (Award Modular v.4.51)**
  - описание формата BIOS'а для его ручной распаковки (на русском языке): http://www.winsov.ru/bios002.php;
- **AMD 762 Chipset Tweaking (MP/MPX) Guide**
  - статья по редактированию регистров чипсета (на русском языке): http://www.tweakfactor.com/articles/tweaks/amd762/1.html;
- **Using Oda's WPCREDIT On VIA Motherboards**
  - разгон системы путем редактирования регистров чипеста (на английском языке): http://www.overclockers.com/tips105/index.asp;
- **H.Oda's Home Page**
  - утилита для модификации регистров чипеста на лету: http://www.h-oda.com/;
- **Award BIOS Editor**
  - редактор Award BIOS'ов, распространяемый с исходными текстами на бесплатной основе: http://awdbedit.sourceforge.net/;
- **AWDhack v1.3**
  - утилита для автоматизированного внедрения своего кода в BIOS http://webzoom.freewebs.com/tmod/Awdhack.zip;