завоевание Интернета или кто сказал "мяу"

крис касперски ака мыщъх, no email

"Я считаю, что должен сделать то, что необходимо — для страны в целом и национальной инфраструктуры, в частности. У меня есть информация, что подрывные элементы уже активно занимаются диверсионной деятельностью против IOS. Я считаю необходимым рассказать всем, что да, IOS уязвима"

Майкл Линн.

дыра, обнаруженная в маршрутизаторах CISCO и обнародованная на хакерской конференции Black Hat 2005 USA, наделала столько шуму, что попала на страницы некомпьютерных газет. о ней много пишут, но все как-то в общих словах никакой конкретики. говорят о скором конце Интернета, пугают захватом управления над магистральными каналами связи, но исходных кодов эксплоита не показывают…

введение или немного предыстории

Все началось с того, что 26 января 2005 года, телекоммуникационный гигант CISCO Systems обнародовал сообщение о дыре в своей новой операционной системе CISCO IOS, установленной на миллионах марштузизаторах ("Cisco Security Advisory: Multiple Crafted IPv6 Packets Cause Reload" http://www.cisco.com/warp/public/707/cisco-sa-20050126-ipv6.pdf). Однако, информация была неполной. Технические детали отсутствовали и добыть их легальным путем не удавалось. CISCO явно что-то скрывала, прячась за туманными фразами, которые можно было трактовать и так, и эдак. Компания ISS (Internet Security Systems), специализирующаяся на информационной безопасности, решилась на собственное расследование. Провести его поручили молодому (всего 24 года), но довольно продвинутому хакеру Майклу Линну (Michael Lynn). Шеф вызвал его к себе на ковер и спросил (маты я опускаю): "Can you reverse-engineer... can you disassemble IOS... to find out what their vulnerability is?" /* *Можешь ли ты дизассемблировать ISO и разобраться с этой уязвимостью?*" */. Ну какой бы хакер ответил нет?

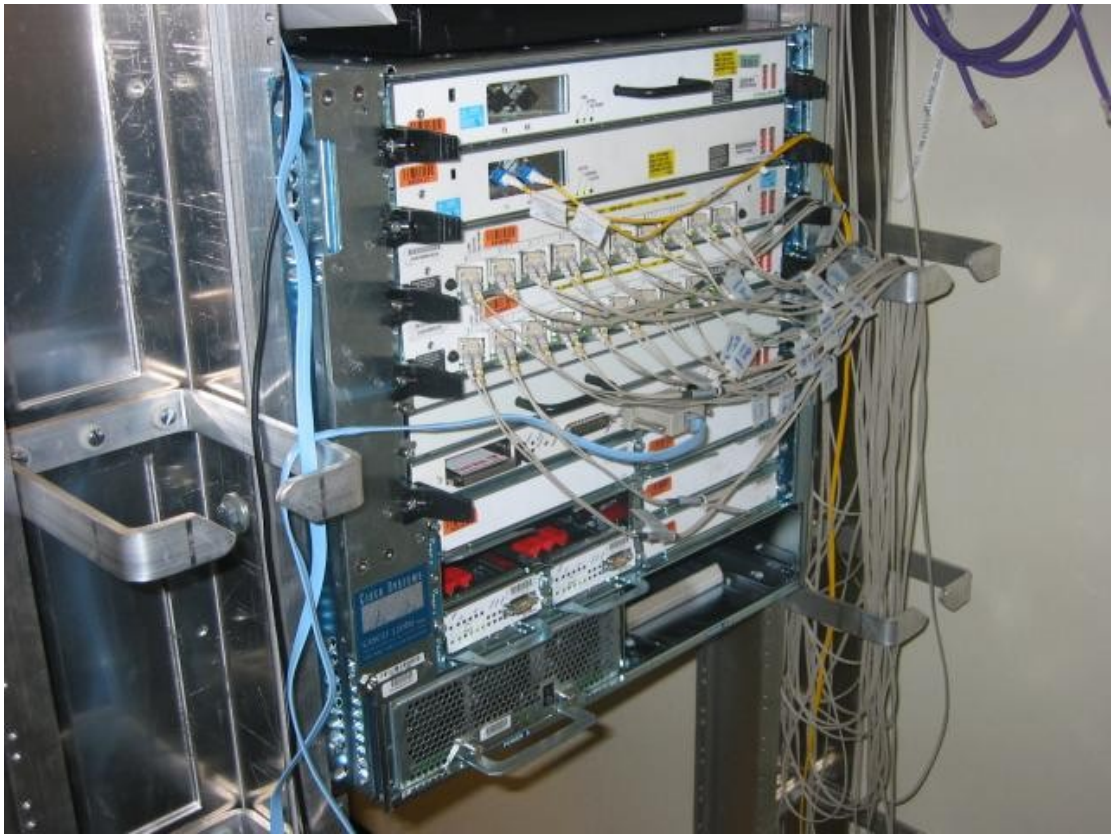


Рисунок 1 киски в боевой стойке

Всю ночь Майкл пил кофе и трахал кисок, но все-таки нашел… совсем другую дыру, намного более коварную и могучую. К в общем-то безобидной перезагрузке (reload) добавился захват управления, а это уже серьезно. Представитель ISS немедленно позвонил в CISCO и сказал: "OK, we aren't 100 percent sure that we found the same bug that you're talking about, but it's important we find out because the one we found has much, much greater impact. You said there's (the possibility) of a denial-of-service attack. But the one we found is fully exploitable" /* *Хорошо, мы на 100% не уверены, что мы нашли тот же самый баг, о котором вы говорили, но наш баг, гораздо более серьезен. Вы говорили, что возможен только отказ в обслуживании, но баг, найденный нами, допускает захват управления*" */, но там не поверили: "You guys are lying. It is impossible to execute shell code on Cisco IOS" /* *Ваш парень лжет. Это невозможно выполнить shell-код на CISCO IOS* */. Майкла снова вызвали на ковер, приказав написать экплоит: "Mike, your new research project is Cisco IOS. Go find out how to exploit bugs on Cisco IOS so we can prove these people wrong" /**Майкл, твой новый исследовательский проект есть CISCO IOS. Вынь да положь рабочий эксплоит для CISCO IOS, чтобы мы могли доказать, что те редиски не правы* */

Рисунок 2 логотип CISCO SYSTEMS

Весь следующий месяц Майкл провел в ожесточенных исследованиях. Но даже имея работоспособный эксплоит на руках, ISS так и не смогла убедить телекоммуникационного гиганта, что его марштутизаторы дырявы как старый галош. Только 14 июня (т. е. спустя три месяца!) они выслали инженера, который охарактеризовал себя как "архитектор IOS", чтобы закрыть этот вопрос раз и навсегда. Майкл в присутствии адвоката продемонстрировал работу эксплоита, натянув маршутизатор по самые помидоры. Это повергло инженера в глубокий шок, но вместе с тем и развеселило: "Wow, that's cool" /* *Вау! Это круто!* */. Инженер ознакомился с черновой версией презентации, которую Майкл планировал продемонстрировать на конференции Black Hat и укатил назад в свою компанию…

Руководство ISS отнеслось к презентации с большим одобрением: "Hey, you want to go to Black Hat? We'd like you to do it" /* *Эй, ты хочешь выступить на Black Hat'е? Это нам нравится!* */ и порекомендовало распространить эксплоит среди всех тестеров компании: "Give this to all the sales engineers and to all the pen testers" /* *Раздай его всем инженерам по продажам и всем бумажным тестерам* */, но Майкл опасался за последствия: "You do realize if you do that, it's going to leak?" /* *Неужели вы не понимаете, что если вы это сделаете, то произойдет утечка?* */. Руководство, недоуменно пожав плечами, возразило: "That's Cisco's problem" /* *Это проблема CISCO* */. Короче все шло своим чередом. Презентация готовилась, а конференция приближалась.

Неожиданно Майкла вздернули на ковер и под угрозой увольнения запретили упоминать факт дизассемблирования IOS. Затем его пригласил на пиво большой босс из CISCO и предложил отложить презентацию… на год. До тех пор пока не будет выпущена новая версия операционной системы. Телекоммуникационный гигант осознавал угрозу, но отчаянно не хотел, чтобы ее осознали другие. Сошлись на том, что вместе с Майклом на сцену поднимается парень из CISCO, который скажет "пару слов", очевидно, обозвав докладчика лжецом, но Майкла это не беспокоило. Чтобы развязать себе руки он уволился из ISS, решив прочитать доклад во чтобы то ни стало. Кто-то же должен предупредить народ об опасности!

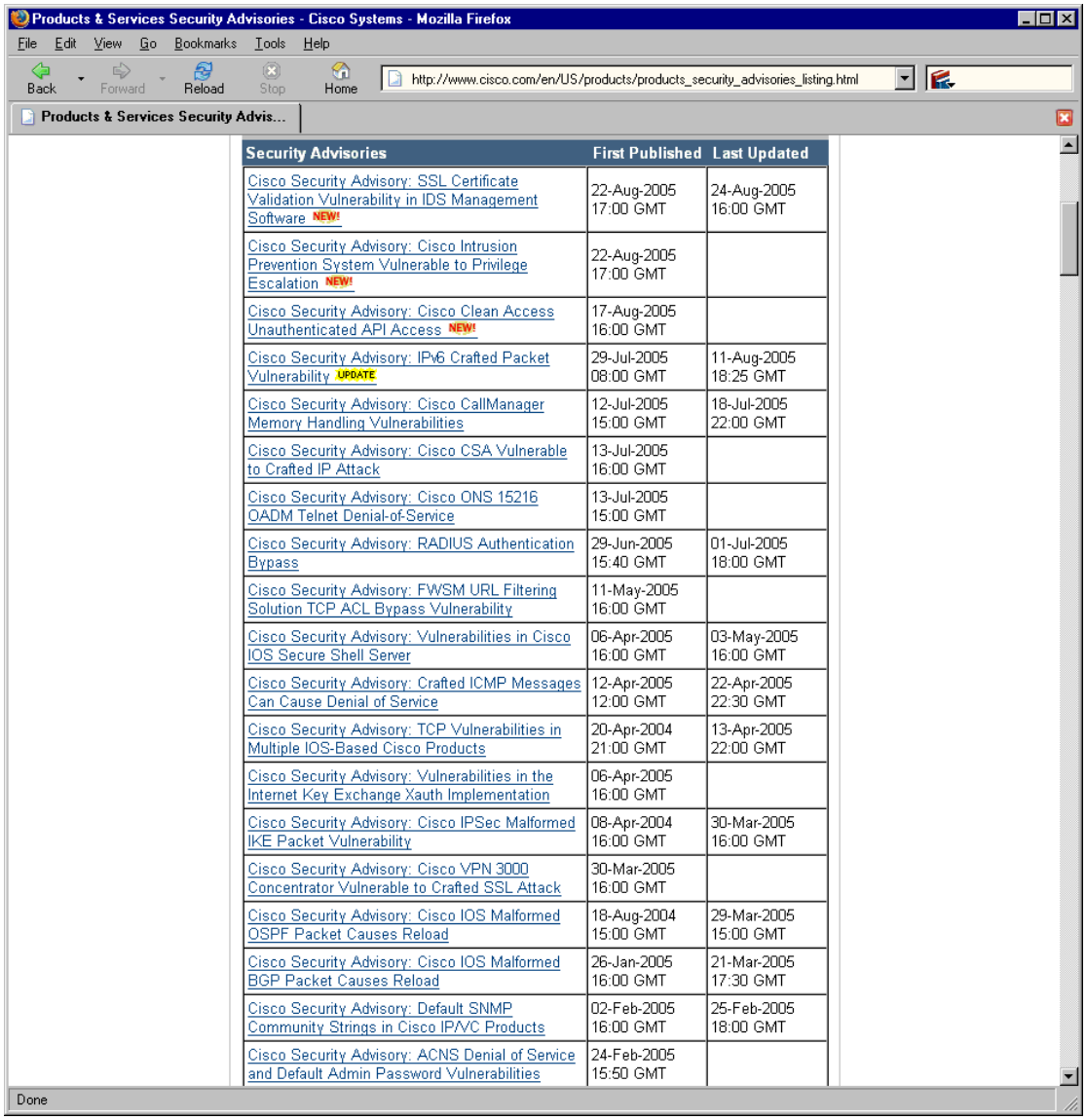
И доклад "The Holy Grail: Cisco IOS Shellcode and Remote Execution" был действительно прочитан! Эффект разодрал аудиторию взрывом атомной бомбы. Майклом заинтересовались Военно-воздушные силы, Агентство Национальной Безопасности и конечно же, небезызвестный CERT. Они предложили ему подключиться к проекту по разработке анти-хакерской стратегии выхода из ситуации, но это уже другая история. Вернемся к CISCO, чья реакция оказалась весьма неоднозначной. Во-первых, при содействии организаторов Black Hat она изъяла текст презентации из материалов конференции и конфисковала сопроводительные компакт-диски, заменив их точно такими же, но без доклада. Во-вторых, она обвинила Майкла во всех смертных грехах и в краже интеллектуальной собственности в том числе.

Сейчас Майкла ожидает куча судебных исков и разбирательств, а CISCO ведет охоту на всех тех, кто осмелился выложить копию доклада в Интернет. К счастью, всемирная сеть живет по своим законам и любые попытки взять ее под контроль имеют обратный результат. Копии

плодятся как кролики. Оригинальную презентацию можно скачать, например, здесь: http://www.security.nnov.ru/files/lynn-cisco.pdf, а здесь лежит видео-ролик, запечатлевший конфискацию дисков в самой "демократической" стране: downloads.oreilly.com/make/cisco.mov.

минет или менуэт?

Неверно считать, что до презентации оборудование CISCO считалось неуязвимым. Так думать мог либо некомпетентный специалист, либо маркетоид. Это не первая и не последняя уязвимость в IOS. Дыры в маршрутизаторах обнаруживались и раньше. За последние пять лет их накопилось около двухсот (!), в чем легко убедиться, посетив сайт CISO: http://www.cisco.com/en/US/products/products_security_advisories_listing.html. А ведь это только официально подтвержденные уязвимости! Неподтвержденных, естественно больше.



Security Advisories	First Published	Last Updated
Cisco Security Advisory: SSL Certificate Validation Vulnerability in IDS Management Software NEW!	22-Aug-2005 17:00 GMT	24-Aug-2005 16:00 GMT
Cisco Security Advisory: Cisco Intrusion Prevention System Vulnerable to Privilege Escalation NEW!	22-Aug-2005 17:00 GMT	
Cisco Security Advisory: Cisco Clean Access Unauthenticated API Access NEW!	17-Aug-2005 16:00 GMT	
Cisco Security Advisory: IPv6 Crafted Packet Vulnerability UPDATE	29-Jul-2005 08:00 GMT	11-Aug-2005 18:25 GMT
Cisco Security Advisory: Cisco CallManager Memory Handling Vulnerabilities	12-Jul-2005 15:00 GMT	18-Jul-2005 22:00 GMT
Cisco Security Advisory: Cisco CSA Vulnerable to Crafted IP Attack	13-Jul-2005 16:00 GMT	
Cisco Security Advisory: Cisco ONS 15216 OADM Telnet Denial-of-Service	13-Jul-2005 15:00 GMT	
Cisco Security Advisory: RADIUS Authentication Bypass	29-Jun-2005 15:40 GMT	01-Jul-2005 18:00 GMT
Cisco Security Advisory: FWSM URL Filtering Solution TCP ACL Bypass Vulnerability	11-May-2005 16:00 GMT	
Cisco Security Advisory: Vulnerabilities in Cisco IOS Secure Shell Server	06-Apr-2005 16:00 GMT	03-May-2005 16:00 GMT
Cisco Security Advisory: Crafted ICMP Messages Can Cause Denial of Service	12-Apr-2005 12:00 GMT	22-Apr-2005 22:30 GMT
Cisco Security Advisory: TCP Vulnerabilities in Multiple IOS-Based Cisco Products	20-Apr-2004 21:00 GMT	13-Apr-2005 22:00 GMT
Cisco Security Advisory: Vulnerabilities in the Internet Key Exchange Xauth Implementation	06-Apr-2005 16:00 GMT	
Cisco Security Advisory: Cisco IPSec Malformed IKE Packet Vulnerability	08-Apr-2004 16:00 GMT	30-Mar-2005 16:00 GMT
Cisco Security Advisory: Cisco VPN 3000 Concentrator Vulnerable to Crafted SSL Attack	30-Mar-2005 16:00 GMT	
Cisco Security Advisory: Cisco IOS Malformed OSPF Packet Causes Reload	18-Aug-2004 15:00 GMT	29-Mar-2005 15:00 GMT
Cisco Security Advisory: Cisco IOS Malformed BGP Packet Causes Reload	26-Jan-2005 16:00 GMT	21-Mar-2005 17:30 GMT
Cisco Security Advisory: Default SNMP Community Strings in Cisco IP/VC Products	02-Feb-2005 16:00 GMT	25-Feb-2005 18:00 GMT
Cisco Security Advisory: ACNS Denial of Service and Default Admin Password Vulnerabilities	24-Feb-2005 15:50 GMT	

Рисунок 3 страничка официального сайта CISCO с описанием подтвержденных дыр

Существует множество эксплоитов, в том числе и с переполнением буфера, через который засылается shell-код, берущий маршрутизатор под свой контроль. Их можно найти практически на любом хакерском сайте. Например: http://www.securiteam.com/exploits/5OP0L1FCAE.html, http://www.antiserver.it/Cisco-Exploit/ и т. д. В частности, еще три года назад в CISCO IOS обнаружилось переполнение буфера, приводящее к захвату управления и был написан демонстрационный эксплоит продемонстрированный на конференции Black Hat 2002 Asia ("Attacking Networked Embedded

Systems" http://www.blackhat.com/presentations/bh-asia-02/bh-asia-02-fx.pdf), детально описанный в 60 номере журнала Phrack ("Burning the bridge: Cisco IOS exploits").

Так что заслуги Майкла и масштабы угрозы сильно преувеличены. Он не был первопроходцем. Обнаруженная им уязвимость применима только к IPv6 (он же "Интернет 2"), и только к IP-пакетам, пришедшим с локального интерфейса. То есть, хакнуть свой собственный маршрутизатор можно, а вот чей-то чужой, взятый наугад, уже нет. Вот тебе и власть над магистральными каналами, вот тебе и Интернет, поставленный на колени.

Вопреки распространенному мнению, рабочий код эксплоита ни на конференции, ни в сопроводительных материалах, так и не был продемонстрирован. Майкл не оставил никаких намеков в каком направлении рыть, но это не помешало остальным хакерам повторить его подвиг, и дыра была переоткрыта заново, однако, о каком бы то ни было практическом использовании говорить слишком рано. IPv6 войдет в нашу жизнь не через год, и не через два, а к тому времени IOS будет повсеместно или практически повсеместно обновлена. Впрочем, кое-где IPv6 все-таки используется (особенно у аплинков), так что подходящую "дичь" можно найти всегда.

свет и тьма в конце тоннеля

Дыры в маршрутизаторах — вполне закономерное явление, которого следовало ожидать. Еще ни одному разработчику не удалось реализовать TCP/IP без ошибок. Обнаруженные уязвимости — симптом тяжелой болезни. До сих пор маршрутизаторы работали лишь потому, что выпадали из поля зрения хакеров, которым намного более выгодно ковырять Windows/LINUX/BSD, чем возиться с кисками. Да оно и понятно. Традиционные операционные системы у каждого стоят на столе, а вот до маршрутизатора еще дотянуться нужно! Завладеть такой штукой может далеко не каждый, к тому же дизассемблирование IOS требует высокой квалификации и специальной подготовки. Готовой информации нет никакой и каждый шаг требует кучи исследований. Вместо наезженной дороги перед нами расстилается сумеречная тьма непроходимой местности, усеянной множеством ловушек. Впрочем, не все так сложно. Как говорится, что сделано одним человеком, может быть понято другим. Главное даже не знания. Главное — это желание и настойчивость.

CISCO наступила на грабли. И скоро получит в лоб. Она выиграла тактическое сражение, но проиграла стратегическую войну. Попытка удержать информацию под спудом, породила скандал, а скандал породил интерес. Хакеры всколыхнулись и бросились штурмовать ISO. "*Причина, по которой мы это делаем, заключается в том, что кто-то сказал: вы не сделаете этого*" — сказал один из них. "*Линн не ограничился только идеями, хотя и не сообщил всех деталей. Но он сказал достаточно, чтобы люди могли понять, как им действовать, и они сделали это*" — добавил другой.

Рисунок 4 хакер, повторивший подвиг Майкла

Всплеск интереса к IOS обещает принести множество новых дыр, так что следующий год должен быть весьма "урожайным". Но как подступиться к маршрутизатору? Монитора нет, клавиатуры нет… Какие инструменты нам понадобиться? Какие машинные языки следует изучить? Короче, для начала исследований нам нужен хороший стартовый пинок… э… толчок.

внутри маршрутизатора

Архитектурно киска состоит из материнской платы, процессора, памяти, шины и интерфейса ввода/вывода. Процессоры довольно разнообразны. В зависимости от модели марштутизатора в них может быть установлен и традиционный Intel, и Мотороллер, и MIPS. В частности, на дизассемблерных фрагментах, приведенные в презентации Майкла, легко узнаются PowerPC, так что поклонники x86 отдыхают или в срочном порядке изучают ассемблеры для остальных платформ. Перечень используемых процессоров приведен в таблице 1.

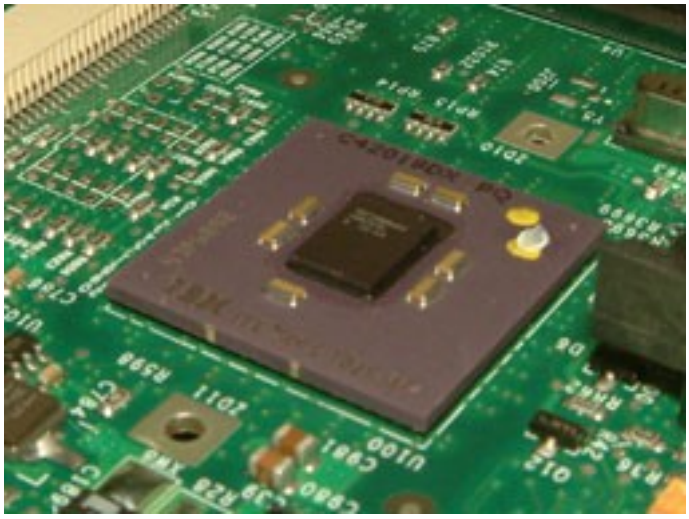


Рисунок 5 процессор PowerPC, установленный в киску

модель	процессор
700	x86 (Intel)
100x, 160x	MC68360 (Motorola)
2500	MC68030 (Motorola)
2600	PowerPC MPC860 (Motorola)
3810	PowerPC MPC860 (Motorola)
3600	MIPS R4700 (IDT)
4000	MC68040 (Motorola)
4500,4700	MIPS R4700 (IDT)
7200	MIPS R4700 (IDT)
7200 NPE 200 и выше	MIPS R5000 (IDT и QED)

Таблица 1 процессоры, использующиеся в машртузаторах CISCO

Полностью укомплектованная киска несет на своем борту четные вида памяти: а) энергонезависимую перезаписываемую FLASH, содержащую сжатый образ операционной системы (для сжатия используется библиотека zlib); б) энергонезависимую перезаписываемую NVRAM со стартовой конфигурацией (startup-config); в) энергозависимую перезаписываемую DRAM/SRAM (обычная оперативная память) и г) энергонезависимую не перезаписываемую постоянную память типа BootROM, содержащую ROMMON код, включающий себя процедуру начального тестирования POST, первичный загрузчик IOS, короче говоря, ПЗУ в обычном его понимании. В усеченных конфигурациях NVRAM может отсутствовать. Подробности можно найти в материале "Cisco Router Forensics" (http://cansecwest.com/core03/CSWcore03-RouterForensics-DDoS-v101.ppt).

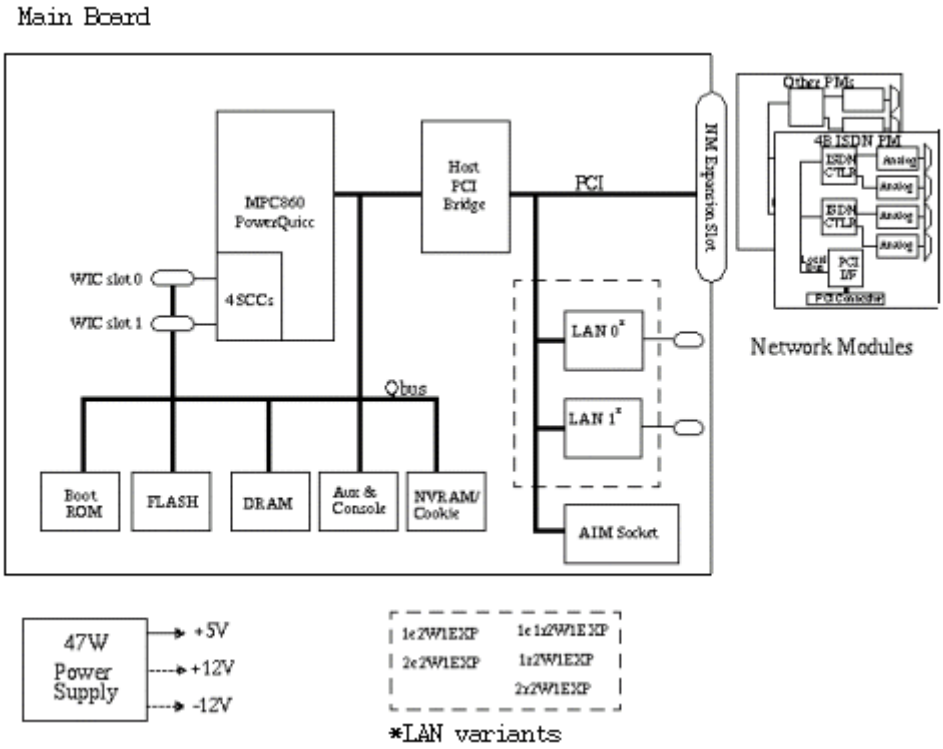


Рисунок 6 схематичное устройство марштутизатора CISCO

Если процессор — сердце маршртутизатора, то операционная система — его душа. В оборудовании CISCO главным образом используются две операционных системы: CatOS и IOS (Internet Operation System), причем последняя намного более популярна. Это операционная система реального времени, скомпилированная gcc и подозрительно похожая на BSD. Она основана на монолитной архитектуре ядра, то есть загружаемых модулей нет, во всяком случае пока. По соображениям быстродействия в ранних версиях оси все процессы работали с одним и

тем же образом (image) и разделяли единое адресное пространство (share memory space). Никакой защиты от воздействий со стороны одного процесса на код/данные, обрабатываемые другим процессом, не предусматривалось, что существенно облегчало написание shell-кода. Так же имелся псевдомногозадачный планировщик не вытесняющего типа "run to completion" (выполнение до завершения). То есть, если в NT операционная система сама переключает потоки без участия со стороны программиста, то в IOS поток должен явно вызвать системную функцию для передачи управления. А это значит, что shell-код может легко захватить власть над системой и не давать себя удалять, впрочем, радоваться по этому поводу слишком рано. Начиная с IOS-XR поддерживается и защита памяти между процессами, и вытесняющая многозадачность. (Подробности о структуре IOS можно почерпнуть из книжки "Inside Cisco IOS software architecture" издательства Cisco Press, которую легко найти в любом парнокопытном).

Поверх ядра накидано множество программного обеспечения, занимающегося самыми разнообразными задачами — от маршрутизации до "чистки" конюшен, причем, в различных "железках" это программное обеспечение сильно неодинаково. И приложения, и ядро работает с одинаковым уровнем привилегий и имеют доступ ко всем системным ресурсам.

Программные файлы представляют собой обыкновенные 32-битные статически слинкованные ELF'ы с покоцанной отладочной инфой (ELF 32-bit MSB executable, statically linked, stripped).

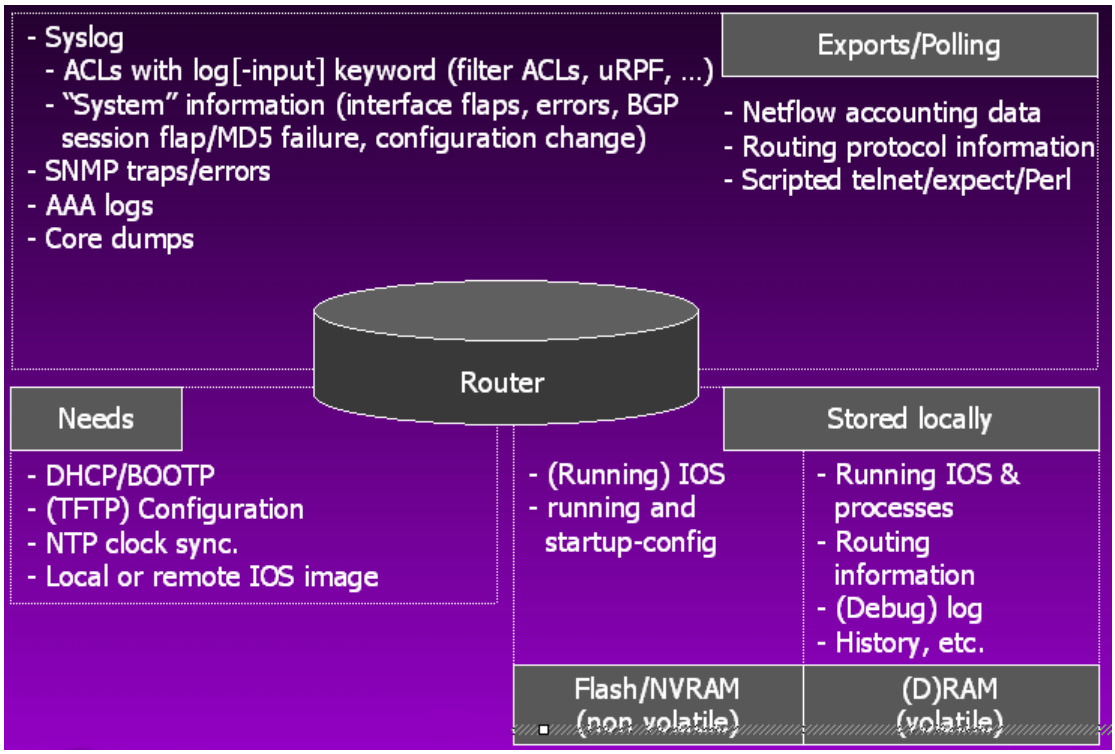


Рисунок 7 структура CISCO IOS

Управление марштузиатором осуществляется через любой внешний порт — от COM-шнурка, до telnet-терминала, работающего на TCP/IP. Интерфейс — командная строка. Среди команд есть как документированные, так и нет (подробнее о недокументированных командах можно узнать у старика Гугла; запрос "undocumented IOS command" выдает тысячи ссылок, среди которых встречается немало полезных, в том числе и http://www.xfocus.net/tools/200307/DOTU.pdf).

Вот, например, результат команды "show proc":

```
scep#show proc
CPU utilization for five seconds: 10%/4%; one minute: 14%; five minutes: 14%
PID QTy        PC Runtime (ms)    Invoked   uSecs    Stacks TTY Process
   1 M*         0         1248        107   11663 2204/4000   1 Virtual Exec
   2 Lst  802DF16        34668        313  110760 1760/2000   0 Check heaps
   3 Cwe  801D5DE            0          1       0 1736/2000   0 Pool Manager
   4 Mst  8058B20            0          2       0 1708/2000   0 Timers
   5 Lwe  80BFD4A           24         46     521 1448/2000   0 ARP Input
   6 Mwe  81F78F0            4          1    4000 1744/2000   0 SERIAL A'detect
```

```
  7 Lwe  80D935A          4         1    4000 1656/2000   0 Probe Input
  8 Mwe  80D8CD6          0         1       0 1744/2000   0 RARP Input
  9 Hwe  80CA966         80        89     898 3116/4000   0 IP Input
 10 Mwe  80F41BA         16       322      49 1348/2000   0 TCP Timer
 11 Lwe  80F5EB8          8         3    2666 3244/4000   0 TCP Protocols
 12 Mwe  813785E         80       177     451 1588/2000   0 CDP Protocol
 13 Mwe  80D5770          0         1       0 1620/2000   0 BOOTP Server
 14 Mwe  81112C0       1356      1522     890 1592/2000   0 IP Background
 15 Lsi  8121298          0        25       0 1792/2000   0 IP Cache Ager
 16 Cwe  80237BE          0         1       0 1748/2000   0 Critical Bkgnd
 17 Mwe  802365A         12         5    2400 1476/2000   0 Net Background
 18 Lwe  804E82E         16         4    4000 1192/2000   0 Logger
 19 Msp  80456DE         80      1493      53 1728/2000   0 TTY Background
 20 Msp  802345C         20      1494      13 1800/2000   0 Per-Second Jobs
 21 Msp  80233F2         68      1494      45 1488/2000   0 Net Periodic
 22 Hwe  80234DC          4         1    4000 1724/2000   0 Net Input
 23 Msp  8023482        772        25   30880 1800/2000   0 Per-minute Jobs
 24 Lwe  8109834          4         2    2000 3620/4000   0 IP SNMP
 25 Mwe  815CE08          0         1       0 1712/2000   0 SNMP Traps
 26 ME   811805A          0        26       0 1892/2000   0 IP-RT Background
 27 ME   803B0F8         32        11    2909 2760/4000   2 Virtual Exec
```

Листинг 1 просмотр списка процессов с помощью команды show proc

Настоящим подарком для хакеров стала команда "gdb", вызывающая встроенный отладчик и поддерживающая следующие подкоманды:

```
gdb
  debug   PID     /* не реализовано                                */
  examine PID     /* отладка процесса с указанным PID              */
  kernel          /* отладка ядра, работает только с консоли       */
```

Листинг 2 команда gdb и ее подкоманды

Однако, прежде чем использовать отладчик его необходимо скомпилировать. Идем на http://www.gnu.org/software/gdb/download/, берем копию посвежее или, наоборот, постарее (предпочтительно использовать gdb-4.18 как наиболее протестированную) и говорим:

```
mkdir m68k-cisco
../configure --target m68k-cisco
make
```

Листинг 3 компиляция отладчика gdb для платформы m68k-cisco

В результате мы получим двоичный файл для платформы m68k. Для остальных платформ компиляция осуществляется аналогичным образом. Теперь можно начинать отладку! Консоль в это время будет нефункциональна, а весь обмен с отладчиком пойдет через его собственный отладочный протокол, описанный в исходном файле remote.c На маршрутизаторе устанавливается серверная часть отладчика, а на терминале — клиентская. Причем отладка ядра (подкоманда kernel) возможна только с консоли.

Дадим команду "`gdb examine 18`", где "18" – идентификатор отлаживаемого процесса (в данном случае "logger"). Подробнее обо всем этом можно прочитать на сайте команды XFocus: http://www.xfocus.net/articles/200307/583.html) Основные отладочные команды перечислены в таблице 2.

Для отладки желательно иметь символьную информацию, однако, IOS это закрытая система с закрытыми спецификациями (ну, не такими уж и закрытыми учитывая, что это порт BSD, унаследовавший родимые пятна багов в zlib, ssh и SNMP) и символьной информации не достать (во всяком случае через легальные каналы), тем не менее корпеть над дизассемблированием дампа не придется.

В мае 2004 года корпоративная сеть CISCO Systems была взломана и исходные тексты системы IOS 12.3, 12.3t попали в руки хакера по кличке franz, который распространил через IRC небольшую часть сорцов ~2.5 Мб в качестве доказательства (самое интересное, что именно в этой месте Майкл обнаружил бага, подозрение усиливается тем фактом, что за ночь дизассемблировать IOS современно невозможно, а именно столько потребовалось ему на анализ).

К настоящему времени исходные тексты просочились в сеть и теперь их можно найти в любом парнокопытном. Полный объем архива составляет 800 Мб и на Dial-Up'e его перекачка

может занять несколько месяцев, однако, она стоит того! Впрочем, отсутствие исходных текстов — это еще не преграда. IDA Pro в руки и айда.

команда	назначение
g	вывести содержимое регистров процессора на терминал
GXX..XX	запись регистров, каждый байт регистров описывается двумя hex-цифрами. Регистры следуют во внутреннем формате gdb, а байты в регистры — в порядке, диктуемом процессором данного типа
mAA.AA,LLLL	чтение памяти, где AA..AA – адрес ячейки, а LLLL – длина блока
MAA.AA,LLLL:XX..XX	запись памяти, где A..AA – адрес ячейки, LLLL – длина блока, а XX..XX записываемые данные
c	продолжить выполнение программы
cAA..AA	продолжить выполнение программы с указанного адреса
s	выполнить следующую машинную команду и остановится
sAA...AA	выполнить машинную команду с указанного адреса и остановится
?	вывести last signal на терминал

Таблица 2 команды, поддерживаемые серверной частью отладчика gdb

где и как искать дыры

Приемы поиска переполняющихся буферов в CISCO IOS мало чем отличается от других операционных систем, но есть в ней и свои особенности. Стек используется крайне редко, в основном она налегает на кучу. По сообщениям CISCO, разрушение кучи — наиболее распространенный баг ее марштутизаторов. Но вот о том, что причиной разрушения являются переполнения динамических буферов она предпочитает умолчать. Так что дыры есть!

Методика переполнения кучи подробно описана в моей книге "hacker shellcoding uncovered" и в статье "Once upon a free()" из 57 номера Phrack'a. В IOS все блоки памяти объединены в двунаправленный список следующего типа:

```
foo->prev->next = foo->next;
foo->next->prev = foo->prev;
```

Листинг 4 двунаправленный список свободных/занятых блоков в CISCO IOS

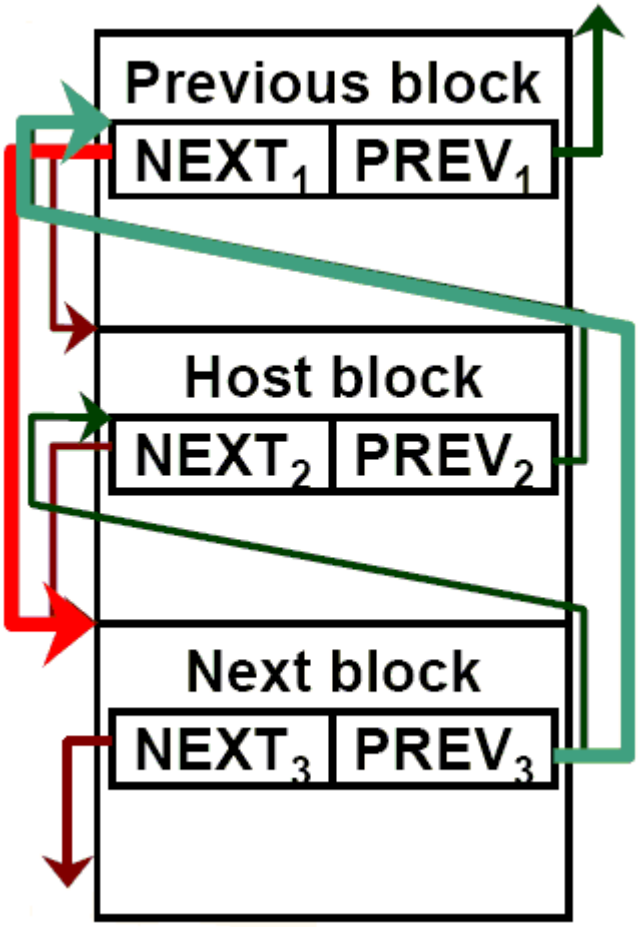


Рисунок 8 организация кучи в CISCO IOS

При освобождении памяти выполняется следующий код, исключающий текущий блок из цепочки занятых блоков:

```
*prev=*next;
*(next+20) = *prev;
```

Листинг 5 псевдокод, освобождения блоки из кучи

Результатом этой операции становится запись в ячейки *prev и *(next+20) значений *next и *prev. Если в результате переполнения нам удастся подменить поля prev и next мы соврем банк, получив возможность писать произвольные данные/код в любое выбранное место. Эта техника (кстати говоря, разработанная хакерами FX и KIMO), получила название "Uncontrolled pointer exchange", но прежде чем ей воспользоваться необходимо познакомится со структурой кучи. Она довольно проста:

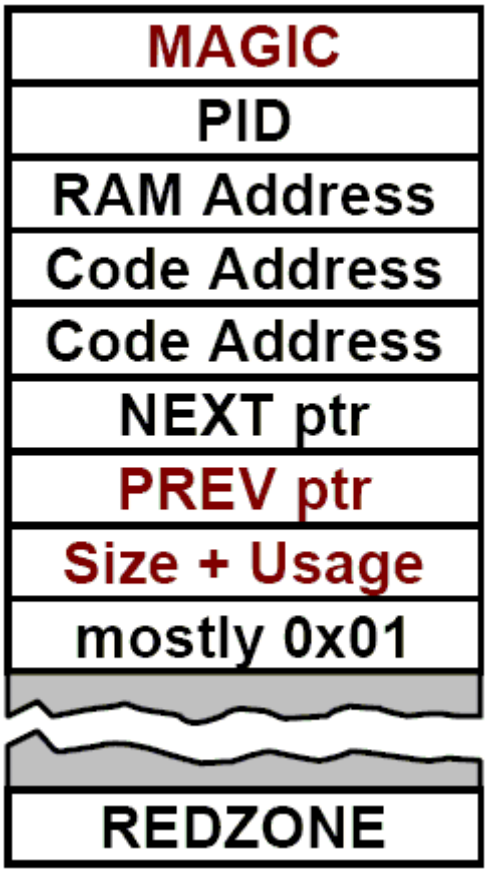


Рисунок 9 структура кучи в CISCO IOS

В начале идет так называемый магический номер (MAGIC), равный AB1234BCh, а в самом конце — Красная Зона (REDZONE), равная FD1001DFh. Обе выполняют охранную функцию и затирать их нельзя. Значение указателя prev проверяется перед освобождением и потому должно быть валидно. В упрощенном виде проверка выглядит так:

```
if (next_block->prev!=this_block+20) abort();
```

Листинг 6 проверка указателя prev перед освобождением

Еще проверяется значение поля Size + Usage, старший бит которого определяет занятость блока (0 — свободен, 1 — занят), что создает проблемы при строковом переполнении, поскольку мы не можем располагать здесь нули и минимальное значение, которое мы можем получить получается слишком большим (7F010101h), однако, тут есть обходной путь. Поскольку, переполнение разрядной сетки никем не контролируется, использование значений типа 7FFFFFFFh дадут ожидаемый результат.

Остальные поля никак не контролируются и могут содержать любые значения. Короче говоря, написание shell-кода вполне возможно. Операционная система IOS использует статические адреса (что есть хорошо), но они меняются от одного билда к другому, а это уже плохо. Поэтому, прежде чем атаковать свою жертву, необходимо тем или иным способом определить версию IOS, иначе червь сдохнет еще в зародыше. Это можно сделать через CDP или SNMP.

Еще хуже, что IOS контролирует целостность кучи и автоматически перезагружает маршрутизатор, если цепочка ссылок (chunk linkage) оказывается разрушенной. За это отвечает специальный фоновой процесс, в зависимости от загрузки маршрутизатора пробуждающиеся каждые 30 или 60 секунд. Именно он проверяет магический номер и красную зону.

Так что shell-коду отпущено совсем немного времени. Конечно, 30 секунд – это целая вечность для процессора, за которую можно не только внедриться в целевое железно, но и заразить множество соседних маршрутизаторов. FX с KIMO предложили несколько решений этой проблемы, но все они оказались нежизнеспособными. Майкл был первым, кому удалось нащупать правильный путь, простой как все гениальное.

Оказалось, процедура abort(), выполняющая перезагрузку, использует специальный флаг-семафор, предотвращающий повторное вхождение (по такому же принципу устроена защита от многократного нажатия Alt-Ctrl-Del в NT). Если мы установим его в единицу, функция abort() тут же выполнит return без всякий перезагрузки. Ниже приведен ее ключевой фрагмент. (Впрочем, марштутизатор все равно будет перезагружен через некоторое время, ведь куча разрушена, так что shell-коду надо спешить).

```
stwu    sp, var_18(sp)
mflr    r0
stmw    r29, 0x18+var_C(sp)
stw     r0, 0x18+arg_4(sp)
lis     r9, (crashing_already_ >> 16)
lwz     r0, (crashing_already_ & 0xFFFF)(r9)
cmpwi   r0, 0
bne     loc_80493D18   # return
```

Листинг 7 фрагмент функции abort

Ни одна из версий IOS, работающих на платформе M68K, не использует аппаратные механизмы контроля за памятью, предоставляемые MMU (Memory Management Unit – Блок Управления Памятью), поэтому запись в кодовый сегмент проходит вполне беспрепятственно. Для защиты от непреднамеренного разрушения используются контрольные суммы. Каждые 30 или 60 секунд специальный процесс сканирует кодовый сегмент на предмет проверки его валидности и перезагружает маршрутизатор, если контрольные суммы не совпадают с расчетными. Однако, против преднамеренной модификации эта "защита" уже не срабатывает, поскольку контрольные суммы легко пересчитать и подправить.

Это можно сделать, например, так:

```
// (c) freedemon
#include <stdio.h>
unsigned char const hexchars[] = "0123456789abcdef";
char tohexchar (unsigned char c)
{
        c &= 0x0f; return(hexchars[c]);
}

int main(int argc, char **argv)
{
        unsigned char checksum; int count; char *command; char ch;
        if (argc <= 1) exit(1); printf("gdb protocol command: ");
        command = argv[1]; putchar ('$');  checksum = count = 0;
        while ((ch = command[count]))
        {
                putchar(ch); checksum += ch; count++;
        }
        putchar('#'); putchar(tohexchar(checksum >> 4));
        putchar(tohexchar(checksum)); putchar('\n');
}
```

Листинг 8 программа для расчета контрольных сумм

На платформе MIPS операционная система IOS ведет себя иначе. На стадии инициализации она перепрограммирует MMU, запрещая модификацию кодового сегмента. Любая попытка записи в эту область вызывает крах системы и последующую перезагрузку маршрутизатора. Против непреднамеренной модификации этот механизм действует очень хорошо, но здравомыслящий хакер, после пятого пива, сможет легко его обойти. Идея заключается в отображении одной из физических кодовых страниц на область данных в записываемый регион. Аналогичный трюк, кстати говоря, используется и для модификации ядра NT. В частности, он применяется во многих брандмауэрах и утилитах Марка Руссиновича.

>>> врезка откуда рыть

Вот три основных источника данных для поиска переполняющихся буферов: исходные тексты, обновления firmware и дамп памяти. Исходные тексты хороши тем, что их легко читать, но они не дают никакой информации о реализации системы на конкретном марштузаторе, к тому же их добыча вообще говоря не совсем законна.

Firmware как правило упакован и перед дизассемблированием его необходимо распаковать, причем сделать это не так-то просто, поскольку CICSO слегка покалечила

заголовок. Майкл использовал WinRAR, другие хакеры используют zlib и пишут свой распаковщик самостоятельно.

Дамп памяти, создаваемый либо по команде, либо в результате краха системы, зачастую бывает сильно поврежден и дизассемблировать его непросто. С другой стороны, он содержит реальный образ рабочей системы и потому наиболее точно отображает текущее положение дел.

заключение

Операционная система IOS чрезвычайно широко распространена: она встречается и в коммутаторах, и в маршртутизаторах, и в точках доступа, однако, не стоит думать, что обнаружив новую дыру мы сможем взять все эти устройства под свой контроль, ведь в них используется различные процессоры и различные версии IOS, а потому с каждой железкой приходится воевать индивидуально. Хорошая новость — в отличии от серверов и персональных компьютеров, далеко не весь парк IOS-оборудования позволяет обновлять прошивку, а даже если и позволяет, далеко не каждый администратор об этом задумывается всерьез.

Рисунок 10 киски — объекты хакерской атаки

>>> врезка кто есть Майкл Линн

Хакер, специализирующийся на встраиваемых (embedded) системах, хачинье ядра, обработке сигналов, криптографии, голосовой телефонии, дизассемблировании и сетевых протоколах. В последнее время сосредоточился на безопасности инфраструктуры критических каналов маршрутизации (securing critical routing infrastructures).

Рисунок 11 Майкл Линн на презентации Black Hat