Today‘s session

- The past
  - Known vulnerabilities
  - Rumors
  - Impact
- The present
  - Heap overflows
  - Stack overflows
  - Shell codes
- The future

The Beginning

- Access List TCP „established“ keyword bug
- First advisory ever published by Cisco, June 2 1995
- Extended access lists where supposed to match TCP packets with ACK and/or RST in them as part of an „established“ connection.
- Bug allowed TCP SYN packets to match this rule
  - Full details never made it to the public (or even PSIRT ?)
  - Apparently, route caching on MCI, SCI and cBus interfaces caused the problem. The caching prevented reevaluation of the ACL rules.

Ref: http://www.netsys.com/firewalls/firewalls-9211/0001.html

The Beginning [2]

- Access List „tacacs-ds“ keyword bug
- July 31 1995
- The keyword was changed from „tacacs-ds“ to „tacacs“
- The command line parser was not backward compatible
  - Extended access lists entries with „tacacs-ds“ were simply ignored.
  - Especially bad for „deny“ rules using the keyword
- Config keyword renaming problems are common on IOS, but this one introduced a security issue

Ref: http://www.cisco.com/warp/public/707/1.html

More IOS bugs

- Cisco PPP CHAP bypass, Oct 1 1997
  - Complete bypass of PPP authentication
  - Details not released, suspected in vendor specific PPP extensions
- „Land“ attack, Dec 10 1997
  - TCP SYN packet with source and destination address and port equal
  - IOS was vulnerable up to the latest version
  - The „new“ CatOS affected as well

Ref: http://www.cisco.com/warp/public/770/land-pub.shtml

More IOS bugs [2]

- Cisco AAA bug, Jan 21 1998
  - Processes using AAA did not receive all answer information from RADIUS/TACACS server
  - If answer contained restrictions, these were not applied
- VTY login bug, Aug 12 1998
  - Bug identified due to customer reports about sporadic router crashes
  - Details not released

More IOS bugs [3]

- The „history bug“, Oct 14 1998
  - „funny“ sequence of characters at the login prompt revealed the command line history of the previous user
  - Cisco advisory talks about a „trusted customer“ finding this in a „lab test“.
- Access Lists again, Nov 5 1998
  - 7k series router distributed fast switching forgets to apply output access list
  - Isn‘t that like the „established“ bug in 1995 ?

More IOS bugs [4]

- The „NAT leak“ bug, Apr 13 1999
  - Packets leaked in NAT config
  - Again said to be found by customer‘s testing
  - The new 12.0 mainline was affected
- ACL „established“ on GSR, June 10 1999
  - Four years later finally found on the 12000 series backbone routers as well
  - Again found by customer

More IOS bugs [5]

- The first HTTP bug - %%, May 14 2000
  - Device freezes when receiving GET request containing %%
  - Can also be triggered differently
- And the next one: HTTP Query, Oct 25 2000
  - Device freezes when receiving GET request for http://device_ip/whatever?/
  - Requires enable password

More IOS bugs [6]

- SNMP all over the place
  - Feb 27 2001, the ILMI community
  - Feb 28 2001, multiple vulnerabilities
    - „cable-docsis“
    - RW community visible via RO walk
    - SNMP trap community valid for RO/RW
- PPTP bug, Jul 12 2001
  - Malformed packet DoS
  - First time Cisco giving creadit (Candi Carrera)

More IOS bugs [7]

- The big HTTP thing, June 27 2001
  - Known as the „exec level“ bug
  - http://<device_addres>/level/**16**/exec/
  - Advisory still saing „no malicious exploitation of this vulnerability is known“.
- Malformed SNMP, Feb 12 2002
  - PROTOS got them all –
    no further comments required

Rumors

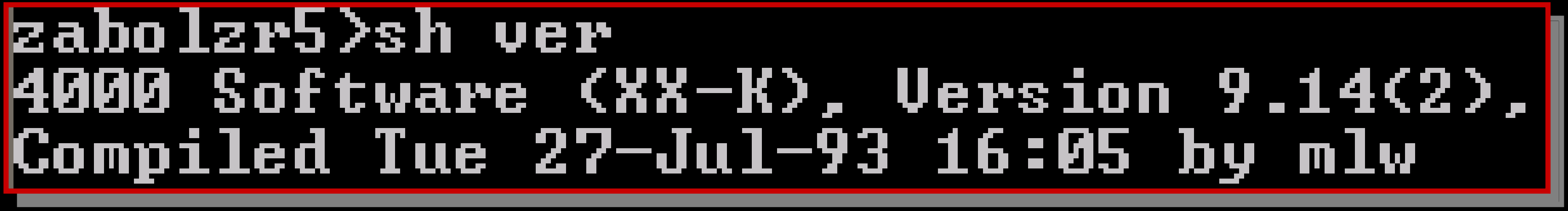
- IOS backdoors
  - Not hard to do
  - /me met people who claimed to have done it
- BGP hack tools
  - ADMbgp exists and seems to work quite well
  - Man-in-the-Middle works fine
- NSA backdoors
  - No indications to that so far

Impact in the past

- Easy Denial of Service conditions frequently exploited
  - Land.c
  - VTY crash bug
- „Death on arrival“ bug
  - Exploitation limited due to core network filters
  - Vulnerable routers will be around for a while
- HTTP „exec level“ bug
  - Widely exploited, no matter what they say
  - Scores of routers owned and reconfigured
  - Guess what, nobody noticed!

The present

Did it get any better?



```
zabolzr5>sh ver
4000 Software (XX-K), Version 9.14(2),
Compiled Tue 27-Jul-93 16:05 by mlw
```

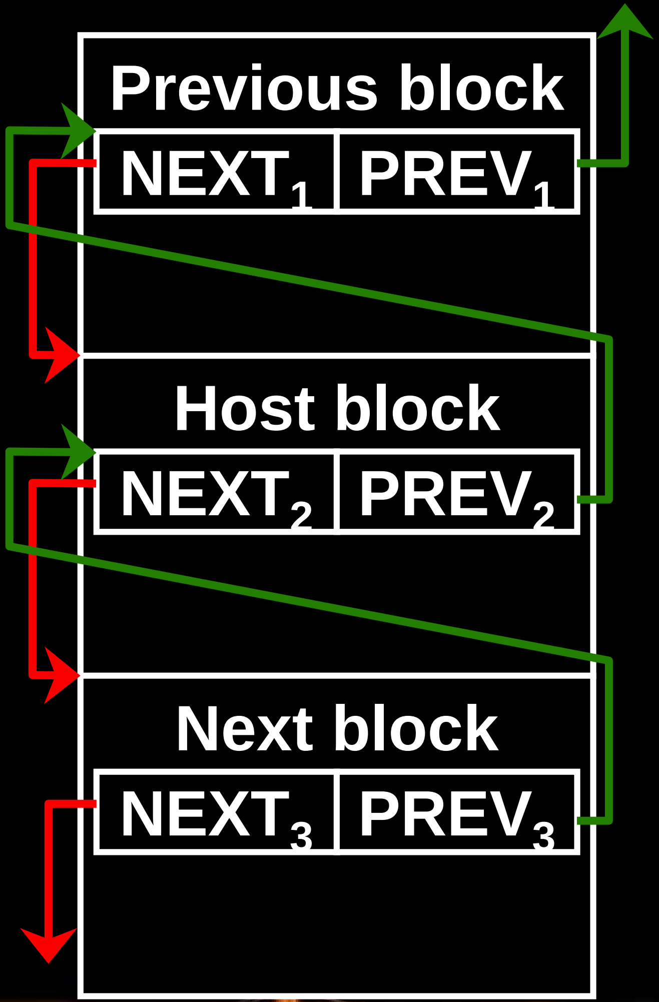
Latest IOS bugs

- Response Time Reporter (or SAA), May 15 2002
  - Single packet DoS
  - Details never released
    - UDP port 1967
    - Data: \x00\x00\x00\x34 + ‚A‘ x 48
- „Death on arrival“ bug, July 16 2003
  - Again a design failure plus bad parsing bug
  - Information released carefully by Cisco to prevent wide spread exploitation

„just“ security notices

- CDP router DoS
- EIGRP router aided network-wide DoS
- TFTP long filename buffer overflow (exploited)
- OSPF buffer overflow (exploited)
- UDP echo service memory leak
- HTTP 2GB request buffer overflow (exploited)

Heap overflows

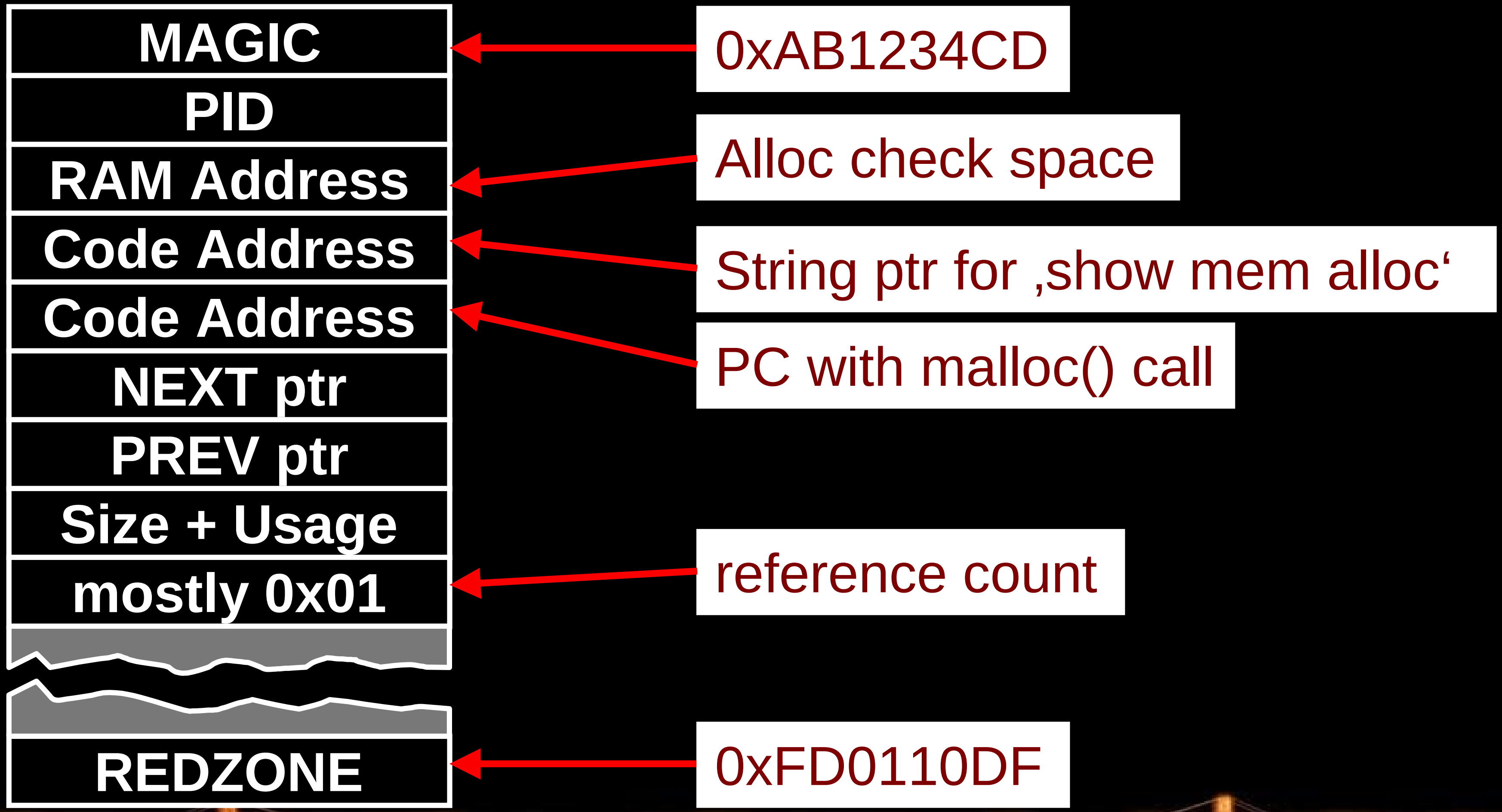


- Two different memory areas: main and IO memory
- Double linked pointer list of memory blocks
  - Same size in IO
  - Various sizes in main
- Probably based off a tree structure
- A single block is part of multiple linked lists

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)

Block layout



MAGIC

PID

RAM Address

Code Address

Code Address

NEXT ptr

PREV ptr

Size + Usage

mostly 0x01

REDZONE

0xAB1234CD

Alloc check space

String ptr for ‚show mem alloc‘

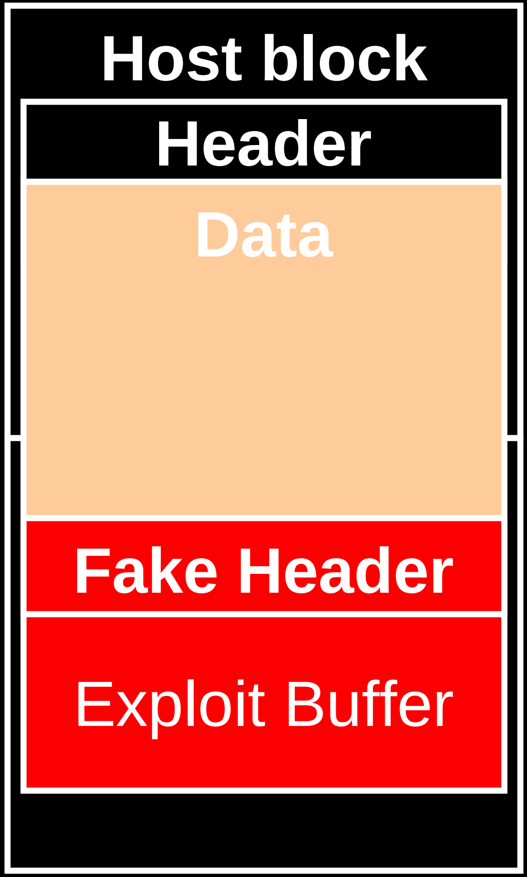
PC with malloc() call

reference count

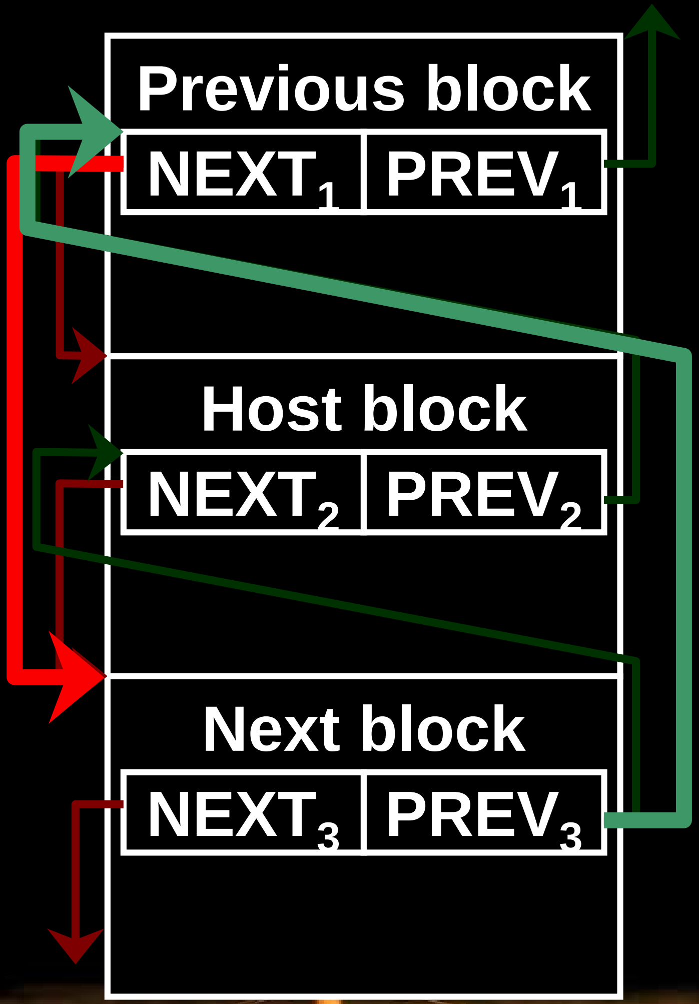
0xFD0110DF

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



Previous block

NEXT$_1$ PREV$_1$

Host block

NEXT$_2$ PREV$_2$

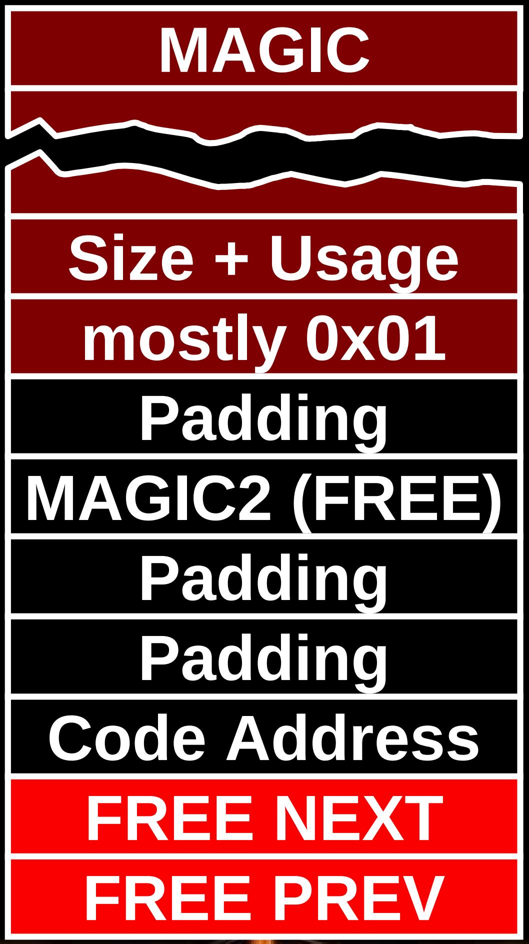
Next block

NEXT$_3$ PREV$_3$

- Remember: 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);
```

Arbitrary Memory write



MAGIC

Size + Usage

mostly 0x01

Padding

MAGIC2 (FREE)

Padding

Padding

Code Address

FREE NEXT

FREE PREV

- 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;
```

Exploitation – issues in the past

- For heap overflows, we need several image and configuration depend addresses
  - PREV pointer in the memory block
  - Size value in IO memory exploitation
  - Stack location
  - Own code location
- Requirements made reliable remote exploitation hard / impossible

A small bug ...

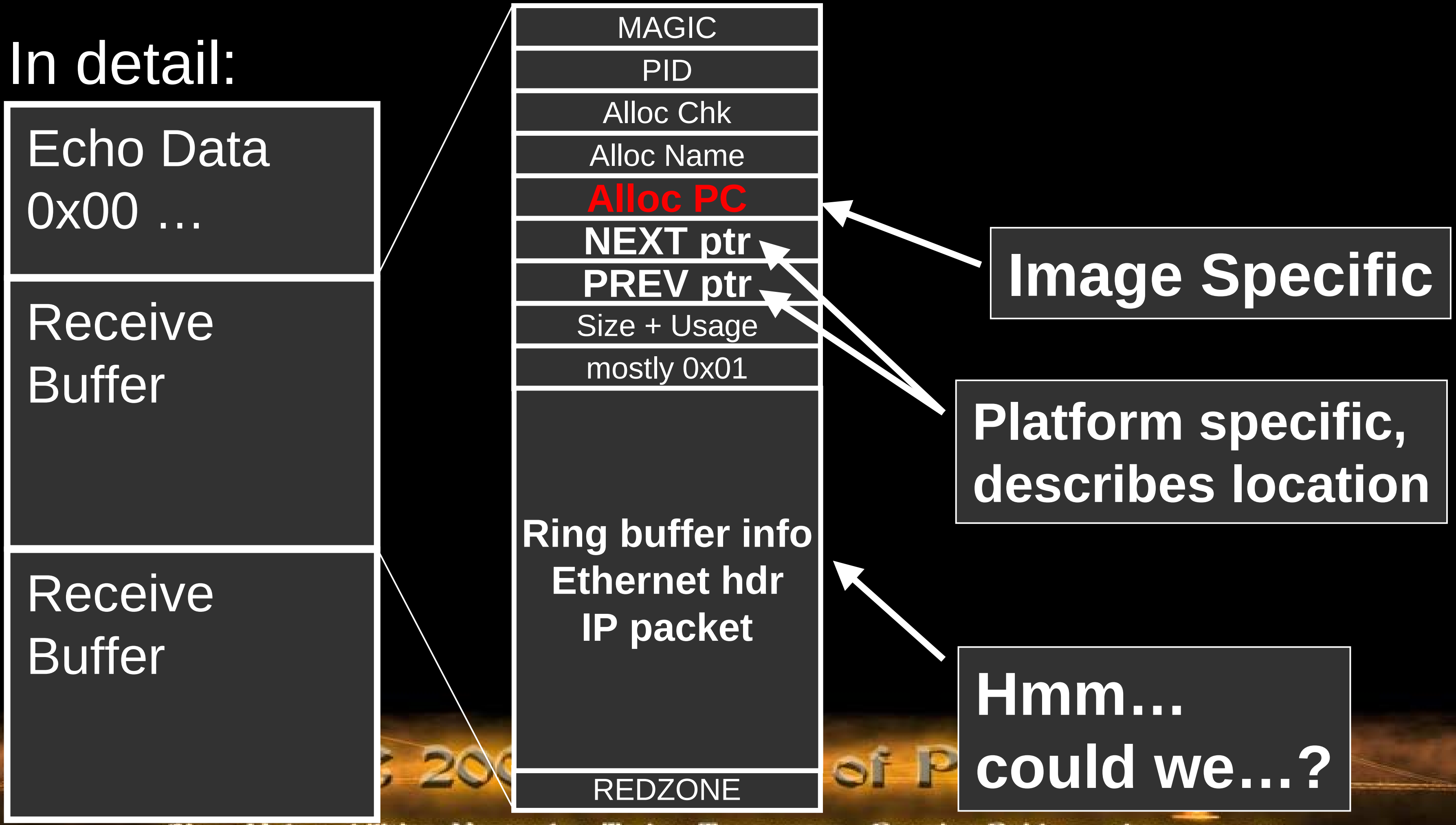
- Cisco IOS 11.x and below
- UDP Echo service memory leak
  - Device sends as much data back to the sender as the UDP length field said it got
  - Leaks IO memory blocks
  - IO memory contains actual packet data – and not just ours
  - We are talking about 19kbytes here
- Comparable bug surfaced in IOS 12.x Cisco Express Forwarding (CEF) code

IOS Fingerprinting

- Leaked IO memory contains memory block headers
  - Block headers contain address of who allocated the block
  - Address of allocating function changes per image
  - Address range changes per platform
- Result:
  Reliable remote IOS fingerprint

IOS Fingerprinting [2]

In detail:



Echo Data
0x00 …

Receive
Buffer

Receive
Buffer

MAGIC

PID

Alloc Chk

Alloc Name

Alloc PC

NEXT ptr

PREV ptr

Size + Usage

mostly 0x01

Ring buffer info
Ethernet hdr
IP packet

REDZONE

Image Specific

Platform specific, describes location

Hmm… could we…?

Remote IOS Sniffing

- Leaked IO memory contains packets in the receive buffers
  (RX ring ds elements)
- Phenoelit IOSniff
  - Repeated memory leak retrieval
  - Memory block identification
  - Packet offset identification
  - Packet decoding
  - Caching and duplicate prevention

```
[0x00E0B42C]: 00:60:47:4F:5E:72 -> 01:00:0C:CC:CC:CC
 pure Ethernet stuff
 .... ....+....radio.b.phenoelit.de......................Ether
 net0............Cisco Internetwork Operating System Software
  .IOS (tm) 1600 Software (C1600-Y-L), Version 11.3(11b), REL
 EASE SOFTWARE (fc1).Copyright (c) 1986-2001 by cisco Systems
 , Inc..Compiled Fri 02-Mar-01 17:12 by cmong....cisco 1603..
---
[0x00E0CF2C]: 00:A0:24:2B:BE:BB -> 00:00:0C:4A:9C:C2
 192.168.1.3 -> 192.168.1.16   43 bytes [TTL 63] DF (payload 23)
 [TCP]  1035 ->    23 (783944042/983338029) ACK PSH  win 32120
(payload 3)
 en.
---
[0x00E112AC]: 00:A0:24:2B:BE:BB -> 00:00:0C:4A:9C:C2
 192.168.1.3 -> 192.168.1.16   46 bytes [TTL 63] DF (payload 26)
 [TCP]  1035 ->    23 (783944045/983338043) ACK PSH  win 32120
(payload 6)
 s3cr3t.
---
[0x00E1196C]: 00:00:0C:4A:9C:C2 -> 00:01:03:8C:9B:44
 [ARP] Reply for 192.168.1.100 from 192.168.1.16 (MAC:
00:01:03:8C:9B:44)
```

IOS HTTP bug

- Almost all embedded HTTP implementations are vulnerable – Cisco is no exception
- Integer or counting related issue
- IOS 11.x – 12.2.x
- Requires sending of a 2GB sized URL to the device
- Stack based buffer overflow

What we got now

- UDP Echo memory leak
  - Attacker provided binary data
    (the delivered Echo content)
  - Live IOS memory addresses
    (leaked IO memory block headers)
  - Ability to fill multiple memory areas with our binary data (Ring buffer)
- HTTP Overflow
  - Direct frame pointer and return address overwrite

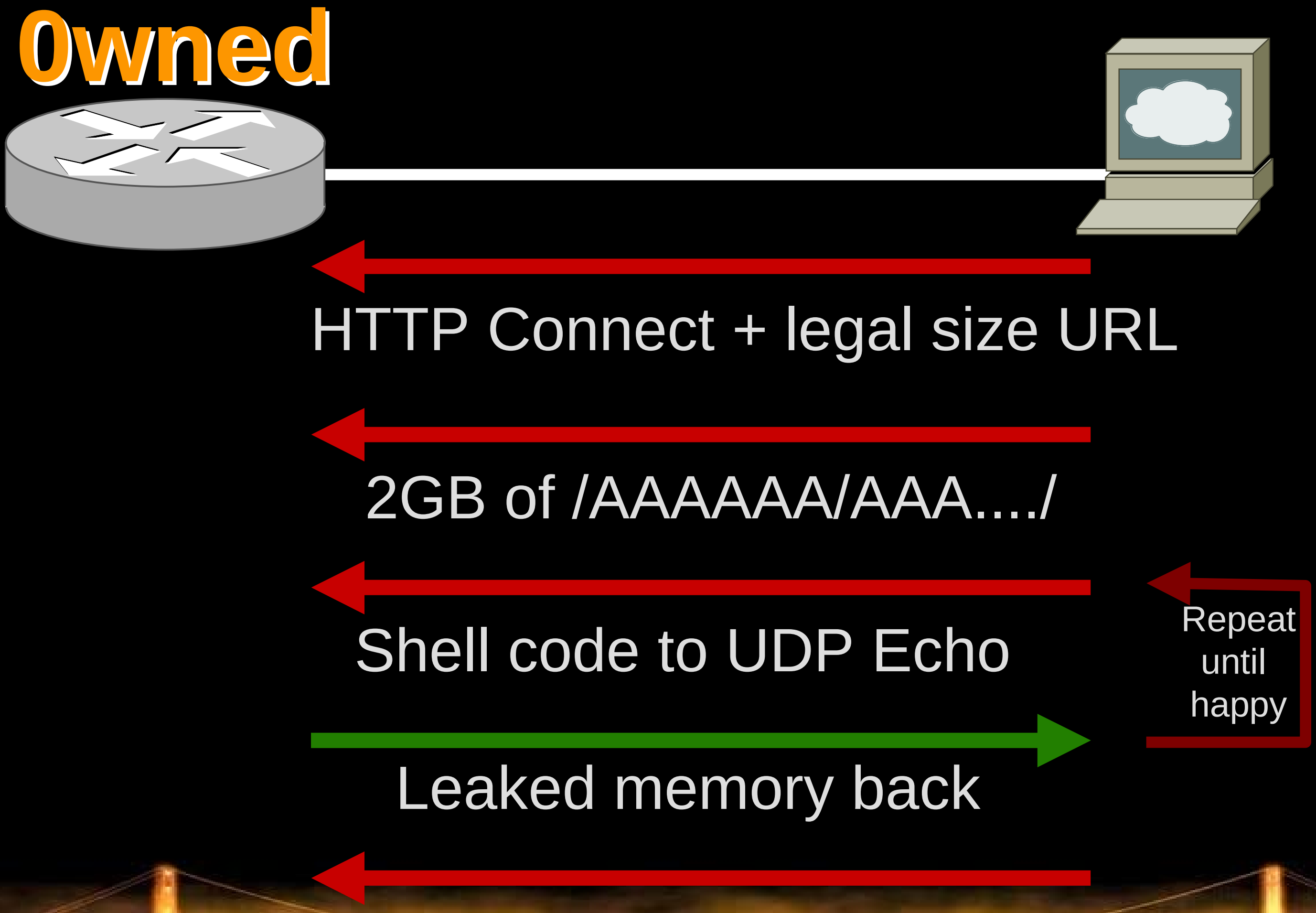
What we can do now

- Send full binary shell code
- Calculate the address of the code using IO memory block header information
- Select the shell code that is most likely not modified
- Directly redirect execution in the provided shell code
- Own the box

Combining

1. Send the maximum URL length allowed by IOS
2. Send 2GB of additional URL elements in correct sized chunks
3. Perform UDP memory leak several times with shell code in the request packet
4. Make intelligent decision on which address to use
5. Complete overflow and gain control

Again, in color



Owned

HTTP Connect + legal size URL

2GB of /AAAAAA/AAA..../

Shell code to UDP Echo

Repeat until happy

Leaked memory back

Complete HTTP overflow

Binary via HTTP

- Cisco‘s HTTP doen‘t like all characters
  - Slash , 0x0a, 0x0d and 0x00 are obviously bad for HTTP
  - Some others are bad as well
- HTTP encoding (%XY) supported
- Decoding seems to take place
  in the exact same buffer
- Return address HTTP encoded

Return address selection

- Several address selection strategies tested
  - Last address obtained
    (about 50% success)
  - Randomly selected address
    (about 50%-60% success)
  - Highest memory location
    (about 0%-10% success)
  - Lowest memory location
    (about 90% success)
  - Most frequently seen address
    (about 30%-40% success)

Researching binary IOS

- Cisco supports serial gdb
- ROM Monitor (rommon) allows limited debugging
  - Breakpoints
  - Watchpoints
  - Disassembly
- Code identification simple
  - Related debug strings can be found in the code
  - Data and text segment are intermixed with each other
  - Strings stored before the related function

Next generation code

- Runtime IOS patching
- Patched (disabled) elements:
    - IOS text segment checksum function
    - Authentication requirement for incoming VTY connections
    - Verification return code from “enable mode” function
    - In the future: ACLs or BGP neighbor check?
- Keep IOS running … but how?

Clean return

- Overflow destroys significant amounts of stack due to HTTP encoding
  - 24 bytes encoded: **`%fe%fe%ba%be%f0%0d%ca%fe`**
  - 8 bytes decoded
- Motorola call structure uses frame pointer in A6 and saved stack pointer on stack
- Moving the stack pointer before the saved SP of any function restores SP and A6
- Search stack “upward” for return address of desired function

```
SP = <current> - 4
unlk a6
rts
```

Clean return code

IOS 11.3(11b) HTTP overflow find-return code

```
    move.l      a7,a2
findret:
    addq.l      #0x01,a7
    cmp.l       #0x0219fcc0,(a7)
    bne         findret
    move.l      a7,(a2)
    sub.l       #0x00000004,(a2)
    move.l      (a2),a6
    clr.l       d0
    movem.l     -4(a6),a2
    unlk a6
    rts
```

Runtime IOS patching

- Advantages
  - Router stays online
  - Configuration preserved
  - Backdoor in IOS runtime code
- Disadvantages
  - Depending on image
  - Large target list required
    (code addresses per image)
  - Annoying “checksum error” message
    on console ☺

CISCO CASUM EST

- Reliable remote IOS exploitation
- Address calculation and shell code placement via UDP Echo info leak
- Address selection using second smallest address
  - first used for HTTP transfer itself
- Runtime IOS patch disables VTY and enable mode password verification

CISCO CASUM EST

```
Trying...
Connected to c1600.mgmt.nsa.gov.
Escape character is '^]'.

radio>en
Password:
Password:
Password:
% Bad secrets

radio#sh ru
Building configuration...
```

A different approach

- Image independent shell code anyone?
  - Modifying IOS code is image dependent
  - Modifying IOS configuration is not
- Runtime config modification code preserves original config and changes only a few „elements“.
- Shell code needs
  - Strstr()
  - Memcpy()
  - Checksum()
- Well, we can do that ☺

Config modification code

- Find beginning of configuration in NVRAM
- Find occurrences of
  `“\n password “`
  `“\nenable “`
- Replace occurrences with your “data”
- Hereby replace authentication information for
  - Console passwords
  - VTY line passwords
  - Enable passwords
  - Enable secrets
- Recalculate checksum
- Reboot

Config modification code

```
nsagw1#sh startup-config
Using 857 out of 7506 bytes
!
version 11.3
service password-encryption
service udp-small-servers
!
hostname nsagw1
!
enable password phenoelit
J5Ct.rs.Ud75tps/nQj0
enable password phenoelit
42410C150C03
!
```

Config modification code

- Advantages
  - Image independent
  - Configuration preserved
  - More choices of what to do
- Disadvantages
  - Depending on platform
  - Router has to reboot once

So what?

- The community gains increasing experience in exploiting IOS bugs
- IOS has still no internal protections
- Features are still added to the old code tree
- IOS still copies data into buffers that are not large enough to hold it
- Note:
  Others exploit IOS as well, only we do it in the public

So what?

“Body of Secrets”, James Bamford:
By looking for vulnerabilities in Cisco Routers, the NSA can find and capture a lot of electronic messages.

NSA Director Terry Thompson:
“But today, I really need someone who knows Cisco routers inside-out and helps me understand how they are used in target networks.

The future

Hope?

They just can‘t parse!

- Most of the bugs discussed are parsing related
- Research indicates that every service process on IOS does it‘s own IP packet parsing
- See yourself:
  - HTTP request:
    ```
    GET / 0x7FFFFFFF.0xFFFFFFFF
    ```
  - Result in debug output:
    ```
    HTTP: client version 2147483647.-1
    ```

Outlook (not Microsoft‘s)

- „Death on arrival“ bug was design related
  - We may see more of these in the future
- Not all overflows are found yet
- A complete rewrite is in progress (again)
  - When will it come and will it be secure?
  - Will it support loadable modules?
  - Third party modules?
- Over 22.000 images in production,
  who is going to update them all?

CISCO SYSTEMS

THIS IS YOUR POWER IN THE NETWORK. now.

FX of Phenoelit

www.phenoelit.de

www.darklab.org

graphics by thg & zet