Developments in Cisco IOS Forensics

Felix ‘FX’ Lindner

BlackHat Briefings
Las Vegas, August 2008
Agenda

- Why Network Equipment Forensics?
- Types of Attacks
- Types of Evidence
- Binary Evidence Analysis
- Reality Check IOS Exploitation
Why Cisco?

- This talk is Cisco centric
  - 92% market share* for routers above $1,500
  - 71% market share* enterprise switch market

- What about Juniper?
  - From both attacker and forensics point of view, Juniper routers are just FreeBSD

- What about <someCheapHomeRouter>
  - From both attacker and forensics point of view, they are just embedded Linux systems

*Source: Randomly stolen
Why Network Equipment Forensics?

• By definition, the goal of computer forensics is to explain the current state of a digital artifact.

• Forensic investigations always consist of
  • Acquisition of evidence
  • Recovering information from evidence
  • Analysis of the information

• For common operating systems, the methods and tools are well established

• For network equipment, they are not
Who would hack routers?

- Compromising one machine
  ... gains you access to one machine.

- Compromising one important machine
  ... gains you access to a couple machines.

- Compromising one switch
  ... gains you access to all machines connected.

- Compromising one router
  ... gains you access to everything in the network.
Who would hack routers?

ARP games blocked by the switch.

Switch separates Hosts

"Behind" the Firewall

Neighbor systems have local firewalls.

BBI
(The Big Bad Internet)
Who would hack routers?

- Separation broken (ARP tricks are transparent now)
- Modification of any traffic
- Hard to recognize from the host

There just is no Reverse-NAC.

BBI
(The Big Bad Internet)

Invent & Verify
Who would hack routers?

BBI (The Big Bad Internet)

- Control over the entire network
- Impersonation of the network against the Internet
And on a larger scale...

The Internet

(OK, maybe this is too large)
One scale down: Network Security

Network Firewall, IDS, IPS

Ingress & Egress Filtering, anti-spoofing, route redistribution

Full Trust within the autonomous system

EIGRP 1
EIGRP 2
EIGRP 3
EIGRP 4
OSPF

Security Labs

Invent & Verify
Network Security

- Network security is hierarchical
  - Defending against your downstream is common
  - Defending against your upstream is rather hard
  - Defending against your peers is rare
- Control anything in the hierarchy and you control everything below
Hierarchical Compromises

Local network compromise
Hierarchical Compromises

Just another router: full control
But we got <secuReProCoToR>

- Secure protocols can guarantee that nobody
  …modified the protocol messages
  …spoofed the communication peer
  …replayed the protocol messages
- But if someone did exactly that, they cannot
do anything about it.
  - The choice is: Availability or Security
  - What would your boss / mom do?

Invent & Verify
But we got <secureProtocol>

If the user could control the path his communication is using, it would be called „source routing“ and there is a reason this is no longer in use anywhere in the Internet: The user would have power over the network.
All this is by design

- In IP networks
  - The network node makes the forwarding decisions
  - The leaf node cannot control the traffic flow
Types of Attacks against Network Equipment

- Protocol based attacks
- Functionality attacks
- Binary exploitation
Protocol attacks

- Injection of control protocol messages into the network (routing protocol attacks)
  - Attacker becomes part of the network’s internal communication
  - Attacker influences how messages are forwarded
- Typical examples include:
  - ARP poisoning
  - DNS poisoning
  - Interior routing protocol injections (OSPF, EIGRP)
  - Exterior routing subnet hijacking (BGP)
### Functionality attacks

- Configuration problems
  - Weak passwords (yes, they are still big)
  - Weak SNMP communities
  - Posting your configuration on Internet forums
- Access check vulnerabilities
  - Cisco’s HTTP level 16++ vulnerability
  - SNMPv3 HMAC verification vulnerability (2008!)
  - `memcpy( MyHMAC, PackHMAC, PackHMAC_len );`
- Debianized SSH keys
- Queuing bugs (Denial of Service)
Binary exploitation

- Router service vulnerabilities:
  - Phenoelit’s TFTP exploit
  - Phenoelit’s HTTP exploit
  - Andy Davis’ FTP exploit

- Router protocol vulnerabilities:
  - Phenoelit’s OSPF exploit
  - Michael Lynn’s IPv6 exploit
Detection and Monitoring

- **SNMP**
  - Polling mechanisms, rarely push messages (traps)

- **Syslog**
  - Free-form push messages

- **Configuration polling**
  - Polling and correlation

- **Route monitoring and looking glasses**
  - Real-time monitoring of route path changes

- **Traffic accounting**
  - Not designed for security monitoring, but can yield valuable information on who does what

*Invent & Verify*
# Who detects what?

<table>
<thead>
<tr>
<th></th>
<th>SNMP</th>
<th>Syslog</th>
<th>Config polling</th>
<th>Route monitoring</th>
<th>Traffic accounting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisioning attacks</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interior routing</td>
<td>Yes</td>
<td>Yes (rare)</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exterior routing</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Illegal access due to config issues</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Access check vulns</td>
<td>-</td>
<td>Yes</td>
<td>Maybe</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Binary exploits</td>
<td>-</td>
<td>-</td>
<td>Maybe (if stupid)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
What do binary exploits do?

- Binary modification of the runtime image
  - Patch user access credential checking (backdoor)
  - Patch logging mechanisms
  - Patch firewall functionality
- Data structure patching
  - Change access levels of VTYs (shells)
  - Bind additional VTYs (Michael Lynn’s attack)
  - Terminate processes
What do binary exploits do?

- Runtime configuration changes
  - Change the running configuration
  - Change settings of state machines (SNMP, etc.)
- Load TCL backdoors
  - Later IOS versions support TCL scripting
  - TCL scripts can bind to TCP ports
  - In some IOS versions, TCL scripts survive VTY termination
Forensics for the Binary Exploit class

What we need:

- Evidence acquisition
- Recovering of information from raw data
- Analysis of information

Plus:

- Good understanding of Cisco IOS internals

Invent & Verify
Cisco IOS Device Memory

- IOS devices start from the ROMMON
  - Loading an IOS image from Flash or network into RAM
  - The image may be self-decompressing
  - The image may contain firmware for additional hardware
- Configuration is loaded as ASCII text from NVRAM or network
  - Parsed on load
  - Mixed with image version dependent defaults of configuration settings
- Everything is kept in RAM
  - Configuration changes have immediate effect
  - Configuration is written back into NVRAM by command
Evidence Acquisition

- Common operating system:
  - Most evidence is non-volatile
  - Imaging the hard-drive is the acquisition method
  - Capturing volatile data is optional

- Cisco IOS:
  - Almost all evidence is volatile
  - What we need is memory imaging
  - On-demand or when the device restarts
  - Restarting is the default behavior on errors!

Invent & Verify
Non-volatile Cisco Evidence

- Flash file system
  - If the attacker modified the IOS image statically
- NVRAM
  - If the attacker modified the configuration and wrote it back into NVRAM
- Both cases are rare for binary exploits
Evidence Acquisition

- Using debugging features for evidence acquisition:
  - IOS can write complete core dump files
  - Dump targets: TFTP (broken), FTP, RCP, Flash
  - Complete dump
    - Includes Main Memory
    - Includes IO Memory
    - Includes PCI Memory
  - Raw dump, perfect evidence
Evidence gathering must be configured beforehand

- Core dumps are enabled by configuration
  - Configuration change has no effect on the router’s operation or performance
- Configure all IOS devices to dump core onto one or more centrally located FTP servers
  - Minimizes required monitoring of devices
  - Preserves evidence
  - Allows crash correlation between different routers
- Why wasn’t it used before?
  - Core dumps were useless, except for Cisco developers and exploit writers
What to do with the core?

- The raw memory dump data must be turned into state information
  - What was going on in the router when the memory dump was taken?
  - What processes handled what data?
  - Where did the data come from?
  - Which packet crashed the router?
Core Dump Analyzer Requirements

- Must be 100% independent
  - No Cisco code
  - No disassembly based analysis
- Must gradually recover abstraction
  - No assumptions about anything
  - Ability to cope with massively corrupted data
- Should not be exploitable itself
  - Preferably not written in C
- As you probably figured out by now, we developed such a tool: Cisco Incident Response (CIR)
Analyzing Cores: Inside Cisco IOS

- One large ELF binary
- Essentially a large, statically linked UNIX program
  - Loaded by ROMMON, a kind-of BIOS
- Runs directly on the router’s main CPU
  - If the CPU provides privilege separation, it will not be used
    - e.g. privilege levels on PPC
  - Virtual Memory Mapping will be used, minimally

Invent & Verify
Processes are rather like threads
- No virtual memory mapping per process
- Run-to-completion, cooperative multitasking
- Interrupt driven handling of critical events
- System-wide global data structures
  - Common heap
  - Very little abstraction around the data structures
  - No way to force abstraction
The Image Blueprint

- The IOS image (ELF file) contains all required information about the memory mapping on the router
  - The image serves as the memory layout blueprint, to be applied to the core files
  - We wish it were as easy as it sounds
- Using a known-to-be-good image also allows verification of the code and read-only data segments
  - Now we can easily and reliably detect runtime patched images

Invent & Verify
Image vs. Core

ELF Header
- Code Segment
- Read-Only Data
- Data

IO Memory
- Code Segment
- Read-Only Data
- Data

BSS data

Invent & Verify
Simple Detections Work Best

Security Labs CIR vs. Topo’s DIK
(at PH-Neutral 0x7d8)

Text Segment Compare

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Offset in ELF</th>
<th>Offset in Core</th>
<th>Length of diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x803B79B4</td>
<td>0x3AFA14</td>
<td>0x3B79B4</td>
<td>4</td>
</tr>
<tr>
<td>0x80CB09A4</td>
<td>0xCA8A04</td>
<td>0xCB09A4</td>
<td>4</td>
</tr>
<tr>
<td>0x80CB0EEC</td>
<td>0xCA8F4C</td>
<td>0xCB0EEC</td>
<td>4</td>
</tr>
</tbody>
</table>

CIR Online case: 120EF269A5BC2320730E60289A4B84D9047CECEEE

Invent & Verify
Heap Reconstruction

- IOS uses one large heap
- The IOS heap contains plenty of meta-data for debugging purposes
  - 40 bytes overhead per heap block in IOS up to 12.3
  - 48 bytes overhead per heap block in IOS 12.4
- Reconstructing the entire heap allows extensive integrity and validity checks
  - Exceeding by far the on-board checks IOS performs during runtime
  - Showing a number of things that would have liked to stay hidden in the shadows 😞
Heap Verification

- Full functionality of “CheckHeaps”
  - Verify the integrity of the allocated and free heap block doubly linked lists
- Find holes in addressable heap
  - Invisible to CheckHeaps
- Identify heap overflow footprints
  - Values not verified by CheckHeaps
  - Heuristics on rarely used fields
- Map heap blocks to referencing processes
- Identify formerly allocated heap blocks
  - Catches memory usage peaks from the recent past

Invent & Verify
Process List

- Extraction of the IOS Process List
  - Identify the processes’ stack block
    - Create individual, per process back-traces
    - Identify return address overwrites
  - Obtain the processes’ scheduling state
  - Obtain the processes’ CPU usage history
  - Obtain the processes’ CPU context

- Almost any post mortem analysis method known can be applied, given the two reconstructed data structures.

Invent & Verify
TCL Backdoor Detection

- We can extract any TCL script “chunk” from the memory dump
  - Currently only rare chunks
  - There is still some reversing to do
  - Potentially, a TCL decompiler will be required
Random Applications

- Find occasional CPU hogs
- Detect Heap fragmentation causes
- Determine what processes were doing
- Finding attacked processes
  - Which process had 200 packets in his hands when he died?
- Research tool
  - Pointer correlation becomes really easy
  - Essential in a shared memory environment
IOS Packet Forwarding Memory

- IOS performs routing either as:
  - Process switching
  - Fast switching
  - Particle systems
  - Hardware accelerated switching

- Except hardware switching, all use IO memory
  - IO memory is written as separate code dump
  - By default, about 6% of the router’s memory is dedicated as IO memory

- In real world installations, it is common to increase the percentage to speed up forwarding

- Hardware switched packets use PCI memory
  - PCI memory is written as separate core dump

Invent & Verify
IO Memory Buffers

- Routing (switching) **ring buffers** are grouped by packet size
  - Small
  - Medium
  - Big
  - Huge
- Interfaces have their own buffers for locally handled traffic
- IOS tries really hard to not copy packets around in memory
- New traffic does not automatically erase older traffic in a linear way
Traffic Extraction

- CIR dumps packets that were process switched by the router from IO memory into a PCAP file
  - Traffic addressed to and from the router itself
  - Traffic that was process switching inspected
    - Access List matching
    - QoS routed traffic

- CIR could dump packets that were forwarded through the router too
  - Reconstruction of packet fragments possible
  - Currently not in focus, but can be done if desired
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>22</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>23</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>24</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>25</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>26</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>27</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>28</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>29</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>30</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>31</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>32</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>33</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>34</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>35</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>36</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>37</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
<tr>
<td>38</td>
<td>0.000000</td>
<td>192.168.2.197</td>
<td>192.168.2.193</td>
<td>Telnet</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Frame 71 (614 bytes on wire, 614 bytes captured)
- Ethernet II, Src: 00:0b:5d:00:68:80, Dst: 00:01:80:8b:00:00,
- Internet Protocol, Src: 192.168.2.197, Dst: 192.168.2.193
- Transmission Control Protocol, Src Port: telnet (23), Dst Port: 50380 (50380), Seq: 4294967295, Ack: 4294967295, Len: 1

File: /Z:/Recuri/Recuri prescribe/teledata/1/queryTag/PacketHeader.pcap 81.8 KB 00:00.00
P135 D135 M0
What about crashinfo?

- Later IOS versions write a text file called "crashinfo" to the flash file system when the router crashes
  - Crashinfo contains fairly little information
  - Contents depend on what IOS thought was the cause of the crash

- We found exploitation cases where the router failed to write core dumps, but did write crashinfo
  - Crashinfo correlation to core dumps will likely become an analysis method in future versions of CIR
State of CIR

- Development of Version 1.0 completed
- Online Service at http://cir.recurity-labs.com
  - Available since February 2008
- Free rootkit detection version available
- Professional version available

- There is a large list of things we want in version 1.1 – feel free to add stuff 😊
Challenges with IOS

- The challenge with IOS is the combinatory explosion of platform, IOS version and additional hardware
- Every IOS image is compiled individually
- Over 100,000 IOS images currently used in the wild (production networks)
  - Around 15,000 officially supported by Cisco
  - Cisco IOS is rarely updated and cannot be patched
- This is a great headache for IOS forensics, but also for IOS exploit writers
Reality Check IOS Exploits

- The entire code is in the image
- Remotely, you have a 1-in-100.000 chance to guess the IOS image (conservative estimate)
- Any exception causes the router to restart
  - This is inherent to a monolithic firmware design, as it loses integrity entirely with a single error
- Stacks are heap blocks
  - Always at different memory addresses
  - Addresses vary even within the same image

Invent & Verify
Reality Check IOS Exploits

- So far, all IOS exploits published use fixed addresses that depend on the exact IOS image being known before the attack
  - IOS’s address diversity is a similar “protection” to the Source Port Randomization patch you applied to your DNS servers recently
- We perform our own research in this area, to make CIR ready for the next generation exploits
- It will most certainly not stay this way!
Let the arms race begin!

<table>
<thead>
<tr>
<th>Next Attack</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rootkit code patching core dump writing</td>
<td>GDB debug protocol memory acquisition</td>
</tr>
<tr>
<td>GDB debugger stub patching</td>
<td>ROMMON privilege mode memory acquisition</td>
</tr>
<tr>
<td>Data segement only backdooring</td>
<td>Data structure validation</td>
</tr>
<tr>
<td>Compiled configuration patching</td>
<td>Configuration de-compilation</td>
</tr>
</tbody>
</table>

Once we get all those Cisco IOS platforms covered, we do pretty good in terms of detection mechanisms. But getting there is a *lot* of work!
Want to learn more?

- We are constantly writing about Cisco IOS related information in the "IOS Crash Analysis and Rootkit Wiki"
- CIR Online is available (registration free)

http://cir.recurity-labs.com/
http://cir.recurity-labs.com/

Felix ´FX´ Lindner
Head
fx@recurity-labs.com

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