Recurity Labs GmbH

Developments in Cisco IOS Forensics

Felix 'FX' Lindner, Recurity Labs

January 2008

Abstract

Cisco System's routers running Cisco IOS are still the prevalent routing platform on the Internet and corporate networks. Their huge population, architectural deficiencies and hugely diverse version distribution make them a valuable target that gains importance as common operating system platforms are closed down and secured.

This paper takes the position that the currently used, well accepted practices for monitoring, debugging and post mortem crash analysis are insufficient to deal with the threat of compromised IOS devices. It sets forth a different method that reduces the requirement for constant logging, favoring on-demand in-depth analysis in case of suspicion or actual device crashes. The paper concludes by presenting the current state in the development of software supporting the proposed method and requesting feedback from the community on the software's future directions.

Introduction

Network devices are at the core of any modern IP network. The design of TCP/IP requires intelligent nodes between networks that decide on a hop-by-hop basis how packets are forwarded from source to destination. If network devices fail, behave unexpectedly or get compromised by attackers, all communication logic depending on the network is endangered.

Today's security protocols can cope with intentional interception and modification of network traffic only in a single way: they terminate the connection or drop the packets. End-to-end communication can currently not go on if any mischief in the network is detected.

Cisco Systems' Internetwork Operating System (IOS) is still one of the, if not the most widely used software, running on core Internet routing infrastructure. Cisco routers and therefore Cisco IOS can be found on virtually any network, many times exclusively.

The ability to perform in-depth analysis and diagnostics on routers running Cisco IOS is therefore critical to ensure security and continuous operation of any network.

A Monolithic Architecture

Cisco IOS is a monolithic operating system. All functionality is linked into a single large binary program at compile time. The result of the linking process is referred to as an IOS image. Since Cisco Systems produced a number of different hardware platforms over the time, individual IOS images were required for each new hardware platform.

Additionally, IOS functionality is grouped into feature sets. Every feature set requires a separate build process, as the feature set decides the amount of hardware resources needed to run the IOS image as well as the pricing of the same.

On a third dimension, Cisco Systems constantly improves, modifies and patches the operating system code base to remove bugs and vulnerabilities. New versions must again be made available to the customer as IOS images.

The early architectural decision of monolithically linked images leads to several tens of thousands different IOS images used in today's networks.

Inside Cisco IOS

An IOS image is in most cases a single ELF file. It is therefore best compared to a very large program on a UNIX operating system platform. However, this program is the entire operating system.

IOS runs directly on the router's hardware. Since no third party software is required or expected on the router, no protection mechanisms exist inside the operating system.

Even if the CPU on a Cisco router supports multiple privilege modes for executable code, IOS will always use the CPU in its most privileged setting. IOS also doesn't use any memory protections beside the most basic write protection of code and read-only data segments, which are applied according to the section information in the ELF header of the image.

Processes on IOS are better described as threads. Every process has full access to the entire memory of the platform. Virtual memory mapping is not used. Scheduling of execution cycles for individual threads is performed using a run-to-completion algorithm, allowing the process monopolize as much CPU cycles as it wants without giving the scheduler a chance to preempt it. Time critical tasks, such as operations on the wire, are implemented using simple interrupt handler routines.

Software Architecture

When the operating system doesn't provide the most basic abstractions and protections, and performance is the most critical property for market acceptance, it leaves very little room for dependable and well structured software architecture.

Cisco IOS is entirely implemented in plain C. Due to the lack of process separation, both in memory and execution time, any fault in the software can cause catastrophic results for the entire system, no matter how insignificant the code is for the operation of the router. Even a programming glitch in something as simple as the command line history impacts availability and security of the entire device.

Programming faults such as uninitialized pointers, buffer overflows or out-of-bounds indexing are common in C programs. On operating systems with clear memory separation and virtual addressing, they usually lead to access into not mapped memory, causing the CPU to raise an exception to the kernel. On shared memory platforms without virtual addressing, such as IOS, the chances of the illegal memory access hitting addresses that are clearly out of bounds are significantly higher. In that case, data is modified that could belong to any functionality of the router, ranging from important to irrelevant.

Attacking faults in software and intentionally triggering the execution of code with input data that causes unintended memory write operations is the heart of remote code execution attacks. The architecture of Cisco IOS makes every part of the operating system code a potential target for such attacks, since everything can write everywhere.

Error Recovery

Given an operating system architecture like IOS, very little can be done once a problem with a process or data structure is identified. A process cannot simply be terminated and restarted when it causes a CPU exception. It would theoretically be possible, but the shared memory architecture leaves no way of knowing what other data structures the process already corrupted before the malfunction became obvious through the exception.

Additionally, the entire heap of IOS is a centrally managed, single large structure. If the heap's consistency is questionable, IOS has no way to repair the heap.

The consequence is that a Cisco router running IOS will force a full device crash once an exception occurred or a data structure, such as the heap, is corrupted. IOS even ships with a process (called CheckHeaps) that traverses the heap data structures and forces a device crash if any of the consistency checks fail.

However, the checks that a router can perform during runtime are limited. After all, the router must dedicate as much resources as possible to forwarding traffic, leaving little room for in-depth verification. The checks performed have to complete fast, otherwise the verification procedure would hog the CPU for a long time due to the scheduling.

Compared to all other possible approaches, automatically rebooting the device and starting over with a freshly loaded IOS is the method that causes the least downtime, an important factor for network operation.

Detecting and Debugging Issues

When a Cisco router running IOS exhibits questionable behavior, the network administrator has only a few options to debug the issue. This assumes that the networking engineer can still access the device via the network or an emergency console connection. If, for example, the router is out of memory and the console session has not been established beforehand, nothing can be done to gain access to the machine and perform debugging.

Assumed an interactive session can be established, the network engineer has a plethora of commands that query and display detailed information at his hands. Unfortunately, those commands are mixed into thousands of other commands that may not be helpful in the particular situation. It is left to the network engineer to know what commands will yield the information that allows him to determine the root cause of the problem – a task not easily accomplished.

The aptly named debug commands are the second option for interactive debugging. They enable specific sections of the IOS code base to output debug information using specialized debug output functions. IOS will make sure that the output actually reaches the console in time and may halt the router’s execution if too many debug messages are produced.

The debugging settings in IOS are even more complex than the informational commands. The network engineer must be able to predict the amount of debugging information produced to prevent the router from halting. He must also know or correctly guess what specific part of the operating system may cause the issue he is investigating to enable the right debugging commands.

The above mentioned procedures all assume the router is still functioning as a whole.

In the common event that the router restarts, either due to an exception or due to a software forced reload after detecting corrupt data structures, almost no debugging can be performed.

Later versions of IOS will write a so-called “crashinfo” file into the flash storage of the router, if there is enough space. The file contains information about some aspects of the router’s state when the crash occurred, including partial stack dumps and fractions of the heap. Unfortunately, the information is a fraction of the entire IOS state at the time of the crash and is biased towards what the rather simple analysis functions of IOS considered the likely cause.

Especially in the event of a targeted attack against a Cisco IOS device, the debugging capabilities and crashinfo files are not sufficient to determine the type of attack and whether it was successful or not. They also don't allow to detect intentionally modified (backdoored) images.

Monitoring

Due to the limited debugging and monitoring capabilities the device itself offers through the command line interface, many networking engineers rely on information obtained via SNMP from their network management system. The NMS will constantly query many different aspects of the router through the Cisco published MIBs, obtaining information comparable to what the information commands on the console have to offer. Correlation of the data is performed on the NMS.

While the SNMP driven approach is a generally well accepted way to deal with networking equipment, it does not offer any advantages for the question of why exactly a device restarted. It also does not help to determine if the restart was caused by a failed attack, part of a successful attack or a functional issue with one of the router's services.

Additionally, SNMP monitoring uses CPU resources, memory and network bandwidth, which could be used for routing.

Security Threat Development

Vulnerabilities in Cisco IOS are as common as with any other functionally rich and widely deployed operating system platform. The architecture of IOS, however, makes exploitation a non-trivial task.

In the past, the common operating systems provided soft enough targets to maintain an ongoing stream of new vulnerabilities that could be used to break into the machines directly. Recently, the major operating system vendors, first and foremost Microsoft, increased the code security significantly. Additionally, exploitation mitigation techniques and OS hardening become the standard on all major platforms.

Cisco IOS therefore moves further into the focus, as the benefits start to measure up to the effort required [3]. Non-publicly operating groups will certainly follow an equivalent path, as infrastructure compromises are still highly rewarding and almost impossible to detect.

Summary

Cisco IOS is still the prevalent router operating system in today's networks. Its architecture and consequently the procedures to debug and analyze it are not suited well for detecting and thoroughly inspecting crash causes, especially intentional attacks.

Cisco Systems recently started to distribute the successor, IOS-XR, which features process separation and the QNX commercial microkernel. However, the extremely large population of IOS devices and the significantly higher hardware requirements of the new IOS-XR limit the impact it has on the currently deployed routing platforms.

Generally, networking engineers are reluctant to move from one image version to another, despite the frequent updates by Cisco Systems. Most production networks stay with two or three minor versions behind the most recent releases, since only older versions provide the reliability they need to operate stable networks.

All the discussed factors lead to a large part of the network infrastructure being vulnerable to attacks and malicious modification, without the appropriate tools to detect and analyze it.

A New Analysis Approach

Based on previous research into the areas of independent runtime debugging capabilities on Cisco routers, Recurity Labs developed a new approach to in-depth analysis of devices running IOS.

At its heart sits the observation that, since IOS is in fact a single process, it should be inspected as such, and not considered an operating system for the purpose of analysis.

Of course, Cisco Systems is fully aware of that fact themselves. Accordingly, IOS ships with functionality that is well known for analyzing misbehaving processes on general purpose operating systems: dumping the entire memory into what is known as core files.

On Cisco IOS, the dump is generally written to a separate machine using standard file transfer protocols. Once the router experiences a CPU exception or detects corrupt memory structures, the regular operation is halted and a very small part of the code handles dumping the memory areas onto the preconfigured destination.

Core Files and Transfer Protocols

Later versions of Cisco IOS will write two core files onto the destination server: the main memory core and the IO memory core. The later contains a dump of the IO memory structures, a region of RAM that is used by the router, among other things, for its packet forwarding functionality.

The core files can be written to different destinations: FTP servers, RCP servers, TFTP servers and the flash disk on the router itself. However, the TFTP method cannot be used with any serious router due to a bug[1] in Cisco IOS' TFTP client implementation that was never fixed.

The ideal place to store the core files is the router's flash file system, since this operating involves the least amount of code. Unfortunately, flash file system space is usually precious on production routers and it may not have enough space for writing two core files. Last but not least, core dumps on the flash file system may go unnoticed.

On the upside of the core writing functionality is the ability to dump the two core files while the router is fully operational. This enables networking engineers to obtain a full snapshot of the entire system's state without interrupting service and endangering availability.

The configuration of IOS to write core files in case system crashes is simple and straight forward.

```
ip ftp username user
ip ftp password password
ip ftp source-interface interf
exception protocol ftp
exception region-size 65536
exception dump ip-address
exception core-file filename
```

[1] The bug, documented as Cisco bug ID CSCds46280, is caused by expecting the next block number to be 65536, while the protocol encodes the block number in 16Bits. [1]

This configuration causes two files stored onto the specified FTP server, named *filename* and *filenameiomem*. A recommended way of naming the file is to use the router's hostname, which allows for easy recognition once a core file is written to the FTP server.

By using the core dump feature, the network engineer receives a full snapshot of the router's state when the crash occurred or whenever he needs detailed internal information. The consistency and amount of data obtained using a core dump exceeds by far the information available in crashinfo files or debug messages before the crash (if any).

Unfortunately, network engineers so far had little use of core dumps, since no software was available to make use of the information contained within them.

A Core Dump Analysis Framework

The situation described above led to the development of a core dump analysis framework, named "Cisco Information Retrieval" or CIR.

To regain the information buried in the raw memory dump files, CIR gradually reconstructs abstraction layers, such as memory layout and internal data structures, to obtain an independent view on the state of the IOS device at the time of writing the core dumps.

The first step is reducing the required information and internally handling the complexity of the problem space of platform, IOS image version and feature set.

For the initial analysis, the software uses the two core files together with a known-to-be-good copy of the IOS image that was supposed to be running on the router in question. The IOS image file is either shipped together with the router on external media (such as a CD-ROM) or the Cisco customer possesses an account to the Cisco Connection Online (CCO) server, where images can be downloaded.

The structural information in the IOS image is used as the initial blueprint of the memory layout. All further analysis is performed by independent code, providing a second opinion view on the information within the core dumps.

Heap Reconstruction

CIR performs a full reconstruction of both the main memory and the IO memory heap structures of IOS. This step allows for a far more complete analysis of the heap's integrity and layout than any of the on-board checks of the router, since it is offloaded to a regular PC and can dedicate significantly more resources to the task.

The heap analysis for example inspects the heap for intentional modifications through exploitation attempts. Heap exploitation methods against IOS always leave footprints in the heap's layout, even if the on-board checks do not recognize them.

Additionally, the IOS heap should pass other tests that are based on its specific design. For example, the doubly linked data structure should never leave any heap region memory unaddressed. Any such "hole" in the heap indicates a severe discrepancy from the intended functionality of the allocation code, likely to be caused by heap overflow exploitation.

Process List

Besides reconstructing the heap data structures, CIR also recovers the IOS process list, including the additional information that is stored for each process.

By reconstructing the individual process states at the time of writing the core dump, CIR enables the network engineer to see into the past of the router's process execution. He can inspect the amount of CPU time individual processes consumed as well as their process stacks.

A large number of correlative analyses methods can be used, only given the fully reconstructed heaps and process information. The methods are equivalent to any advanced debugging techniques on common operating system platforms.

Cisco Router Forensic

The primary purpose of CIR is the ability to perform independent analysis on malfunctioning equipment running Cisco IOS. However, a number of modules in CIR are dedicated towards forensic analysis.

Image Patch Detection

A common attack following successful exploitation of a security vulnerability in IOS code is modifying the running IOS image in a way that allows the attacker further and simplified access to the device. Such modifications include disabling of password verification functions, filter list matching or other security relevant code.

CIR detects modifications of the code segment and the read-only data segment simply by comparing the expected data from the good image with the data in the main memory core of the router. If any modification is detected, the locations in both image and core file are reported, simplifying further inspection of the situation.

TCL Script Extraction

Another common method of installing a backdoor on a Cisco IOS router is the use of TCL scripting.

Recent versions of IOS support TCL as a scripting language for the command line interface as well as to perform VoiceXML tasks. The TCL shell allows binding a script onto a specified TCP port and to execute commands in the authentication context that the script was started with, usually privilege level 15, the highest on IOS.

A range of IOS versions did not terminate such scripts upon termination of the login context the script was started in, resulting in an unexpected backdoor on the router. Cisco Systems fixed the issue in recent releases.

The method itself was known for quite some time used by different people for different purposes, ranging from simplification of operational tasks to maliciously keeping privileged access to equipment. Since its recent disclosure [2], it can be assumed that the method will find further use and refinement by malicious parties throughout the Internet and corporate networks.

CIR is able to identify all TCL scripts loaded onto the router, whether they are part of the IOS image distribution (VoiceXML scripts), loaded from a remote FTP or TFTP server or pasted into a TCL shell session. The scripts are extracted for further analysis.

Traffic Extraction

A large part of the router's main memory is dedicated to packet forwarding tasks. This IO memory contains the packets that need to be forwarded by the router as well as the packets that are destined to the router itself.

The IO memory is on the so-called shared memory platforms organized in ring buffers of different bucket sizes. IOS creates ring buffers for small, medium, large and huge packets. Additionally, configured interfaces have their own ring buffers with a bucket size of the MTU they are servicing.

In a default configuration, about 6% of the router's memory is dedicated to IO memory. Configurations optimized by network engineers often use a far larger percentage to minimize forwarding time.

The IO memory is dumped together with the main memory when a core is written. Given recovered information from the routers' main memory, the data structures pointing to packets in the individual ring buffers are used to find and extract packets.

Due to the fact that the ring buffer usage is not equally distributed but depends on the size of the packets traversing the router, some lesser used buffers contain packets further from the past than others.

CIR uses recovered packet information from the main memory to extract packets from the respective ring buffers and combines them into the PCAP file format, supported by a large number of network analysis tools.

This allows the forensic expert to inspect any traffic that has been passing through the router recently, including traffic that may have caused the router to crash and write the core file.

Advanced Traffic Recovery

Writing core files to an external FTP server is a beneficial method for network engineers managing large router networks. For dedicated forensic activities, this method is not preferable, since the write operation itself will use the ring buffers to store the packets that need to be transferred to the external server.

For such cases, CIR supports a second method of obtaining a snapshot of the router in question. By using the serial line GDB protocol over a local console connection, the router can be placed in a tight debugging code loop inside the IOS kernel.

The serial GDB protocol supports memory read operations over a serial line. Cisco IOS implements this functionality on the standard console port. CIR operates on the live router as it would on a set of core dumps, extracting traffic that has been passing through the router in the recent past.

This method may recover packets that contained exploits or further traffic from an attack. It also allows, given fast response times, to trace spoofed IP traffic back to the originating interface and hereby to one IP hop before the inspected router. Given physical access to the infrastructure, at least parts of the IP routing path can be recovered and the potential sources of the attack limited.

Conclusion

Configuring corporate and carrier networks to use the core dump writing functionality is a viable method, reducing monitoring and logging output and providing far superior in-depth analysis. The method can be applied to crashed as well as operating routers and allows the analysis to take place any time with the appropriate staff available, instead of requiring instantaneous and often unplanned ad-hoc inspection.

The implementation in the form of an independent analysis framework can be geared towards networking engineers as well as forensics experts alike, without the need to decide the application beforehand.

Future Work

The architecture of CIR and the ability to recover all important data structures from router memory can be extended in many different directions.

Recurity Labs currently seeks feedback from the community of networking engineers as well as forensic experts regarding the inspection capabilities they desire most.

The following list of potential functionality is therefore meant as suggestion and inspiration for the community:

- Interface tables and states
- Routing tables
- Router process state (EIGRP, BGP, OSPF, etc.)
- VLAN tables
- ARP tables
- VPN context information, including key material and peers
- Dialer tables
- CDP tables
- Spanning tree data
- CEF tree
- User sessions
- Listening port information and handler process identification
- Configuration recovery

Suggestions for any of the above mentioned or entirely different features, as well as criticism and any other feedback are much appreciated. Please send an email to cir@recurity-labs.com.

References

1. IOS TFTP client bug field notice, http://www.cisco.com/warp/public/770/fn13026.shtml
2. “Creating backdoors in Cisco IOS using Tcl”, Information Risk Management Plc, http://www.irmplc.com/download_pdf.php?src=Creating_Backdoors_in_Cisco_IOS_using_Tcl.pdf&force=yes
3. Information Risk Management Plc, http://www.irmplc.com/index.php/153-Research_&_Development-Embedded_Systems_Security#cisco_research