Vulnerability Mining of Cisco Router Based on Fuzzing

Fengjiao Li, Luyong Zhang, Dianjun Chen
Key Laboratory of Universal Wireless Communications, Ministry of Education,
Beijing University of Posts and Telecommunications, Beijing 100876, China

***Abstract*—Router security analysis plays a vital role in maintaining network security. However, IOS, which runs in Cisco routers, has been proved carrying serious security risks. And in order to improve security, we need to conduct vulnerability mining on IOS. Currently, Fuzzing, as a simple and effective automated test technology, is widely used in vulnerability discovery. In this paper, we introduce a novel testing framework for Cisco routers. Based on this framework, we first generate test cases with Semi-valid Fuzzing Test Cases Generator (SFTCG), which considerably improves the test effectiveness and code coverage. After that, we develop a new Fuzzer based on SFTCG and then emulate Cisco router in Dynamips, which makes it easy to interact with GDB or IDA Pro for debugging. In order to supervise the Target, we employ a Monitor Module to check the status of the router regularly. Finally, through the experiment on ICMP protocol in IOS, we find the released vulnerabilities of Ping of Death and Denial of Service, which demonstrates the effectiveness of our proposed Fuzzer.**

***Index Terms*—Fuzzing, Cisco IOS, Vulnerability, SFTCG**

I. Introduction

With the increasing popularity of network applications in various fields, network security issues become more complex and diverse, and it has attracted much attention from government, industrial and academia.

As the crucial interconnection equipment in networking, router security affects the whole network security seriously. Generally speaking, all the attacks go through a router, but some typical attacks implement just by making use of design flaws of the router, and even worse, attackers finish a direct attack on the router. Therefore, it is necessary to research the vulnerability of the router itself. The router vulnerabilities released is becoming more and more, and according to U.S. National Vulnerability Database (NVD), there had been 1807 Cisco vulnerabilities by April 30, 2014, which makes a serious threat to network security.

For many years in the past, predecessors have done a lot of research, including vulnerability detecting of network protocols based on Fuzzing, Cisco IOS security analysis, Cisco IOS vulnerability exploit and so on. However, there remain some problems unsolved. Firstly, some previous studies focused on the exploitation of vulnerability or router debugging, but the research on discovering vulnerabilities is still limited. Secondly, although Fuzzing has been widely used to detect insecurity of network protocols effectively, Fuzzing framework in previous work cannot be used to test router protocols directly, due to the particularity of routers. Finally, Cisco IOS, which is embedded in Cisco routers, is not open-source. Other than Intel, it adopts MIPS or PowerPC architecture, which makes it hard to refer to existing data.

To address these problems, in this paper, we propose a novel testing framework. Based on it, we design a special Fuzzer to discover the vulnerabilities of parsing protocols in Cisco IOS. The main contributions of this paper are shown as follows.

- We introduce a novel framework for security testing of Cisco routers, including Fuzzer, target simulation, Monitor Module, etc.
- We develop a particular Fuzzer based on STFCG model, significantly improving test effectiveness and code coverage.
- We simulate Cisco router in Dynamips, which provides testing environment and also contributes to debug IOS.
- We employ a Monitor Module to supervise the Target, i.e., the Cisco router. Thus, we can check the status of the router regularly.
- We implement our proposed testing framework and perform experiments to mine vulnerabilities on ICMP protocols in IOS, which validate our proposed framework.

II. Related Work

In recent years, Fuzzing test, as a popular black-box testing technique, is widely used in vulnerability discovery, mainly for applications, files and protocols, but few for routers. The critical step of Fuzzing testing is test-cases generation, which has experienced some improvement. Originally, the test-suit formed base on generation or mutation [1], and it depends partly on the target to be tested. Then, Fuzzing test of block-based test-cases proposed for security testing of protocols [2], which is also adopted in SFTCG model in this paper. This method is widely used in the situation where the tester doesnt know the specification completely of a certain protocol, and it also improves development speed and portability. Later, multi-dimension Fuzzing is put forward to mine more vulnerabilities, some of which can hardly be detected by single-dimension Fuzzing, but it also bring combination blast. To solve this problem, reference [3] generate test-cases directly by dynamic tests on vulnerable function to trigger potential vulnerabilities in the target program. However, the method needs to make a static analysis on the source code, which is inapplicable to test routers. In addition, in-memory Fuzzing test may after all be accepted as a direct and efficient method to discover vulnerabilities of network protocols, but it cant be effectively applied to router protocols for the special nature of the router itself.

978-1-4799-5458-2

Currently, there is few studies on Cisco IOS at home and abroad as for its non-open source. The earliest research, having great influence, is from Lynn, and he proposed, in 2005, a way to bypass IOS CheckHeaps protection mechanism [4]. Then in 2008-2009, Felix studied on overflow attacks towards IOS buffer, virus infection, and malicious code (Shellcode), and achieved remote control of a router. All these are exploiting vulnerability of IOS. Recently, in 2011, Muniz and Ortega, released GDB support for the Dynamips IOS emulator, and represented Fuzzing attacks against IOS [5], which promotes our Fuzzing test in this paper.

Over the past years, many Fuzzing tools, such as SPIKE, Peach, Sulley and so on, have been discovered and also well used in software and protocols tests. However, the automated test tools cannot be directly applied for router protocols and each has its weakness.

III. Preliminaries

A. Fuzzing Test

Fuzzing test, generally used in industry testing currently, is a kind of automated black-box testing technology, and it detects flaws of target by providing them unexpected inputs and monitors exception status of the target. In the process, tester sends malformed invasive data, as the input of software or protocol target, to trigger some kinds of vulnerabilities of target, and at the same time, the monitor captures any exception and records run-status in the implementation of the target program. Especially, it should be able to show memory and register information while the target program throws an exception or crashes, so that tester can mine a wide range of overflow vulnerabilities.

Fuzz testing process consists of a few basic stages as follows: 1) Identifying Target; 2) Identifying Input Vectors; 3) Generating Fuzzing test cases; 4) Executing Fuzzing test cases on Target; 5) Monitoring exception of Target and 6) Analysis exception and identify available exploits [1].

Among all the stages, the most critical step is to build the test suit, which consists of a large number of semi-valid test-cases. The so-called semi-valid data are that the necessary identifies of document and most of the data is valid, while the rest of the data is invalid. When the target application handles invalid data, it may cause the application to crash or trigger a security vulnerability. That is to say, they dont only meet the constraints of the program in order not to be discarded, but also must be malformed to trigger crash.

B. SFTCG Model

SFTCG, an extension of TFTCG [6], is used generate test-cases. The following is a detailed description:

$$\begin{cases} SFTCG = (V, S, G, M, MD, Op, Result) \\ Op = fuzz\_g, single\_m, multi\_m, con\_s \\ Result = medium - case, test - cases, testsuit \end{cases} \quad (1)$$

where, $V = v_1, v_2, \cdots, v_n, 1 \leq i \leq n$ is the vulnerable field set and v_i denotes a weak field in network protocols. $S = s_1, s_2, \cdots, s_m$ is a primitive set of the data samples which are linearly independent and s_i is the ith data sample. The code coverage is high, and $test-case$ is hardly redundant. G is the matrix of generation algorithm and can be computed as $G = (g_{ij})_{n\times p}$, where $g_{ij} = 0$ or $1, 1 \leq i \leq n, 1 \leq j \leq p$. When $g_{ij} = 1$, it says that we mutate v_i field according to M_j, 0 not. $M = (m_{ij})_{n\times q}$ is a matrix of mutation algorithm, where $m_{ij} = 0$ or $1, 1 \leq i \leq n, 1 \leq j \leq q$. When $m_{ij} = 1$, it means that we mutate v_i fields of s_j, 0 not. $MDB = M_1, M_2, \ldots, M_r, (1 \leq i \leq r)$ is a database of malformed data, consisting of boundary value, overlong string, separators, formatting characters, etc., where M_i denotes a type of malformed data in database MDB.

The algorithm that *SFTCG* uses to generate test cases is shown in Algorithm 1.

Algorithm 1 Generate test cases using STFTCG

Input: F, MDB, S
Output: $Testsuite$

```
calculate matrix G, M and D;
Testsuite={};
for each g_ij in G do
    mediucases1 = fuzz_g(v_i, m_j, g_ij);
    testcases1 = con_s(mediucases1);
    add testcases1 into Testsuite;
end for
for each m_ij in M do
    if m_ij = 1 then
        mediucases2 = single_m(s_i, v_j);
        testcases2 = con_s(mediucases2);
        add testcases2 into Testsuite;
    end if
end for
for each testcase in testcases1 and testcases2 do
    mediucases3 = mutil_m(testcase);
    testcases3 = con_s(mediucases3);
    add testcases3 into Testsuite;
end for
```

C. Cisco IOS

In this section, we mainly talk about memory implementation mechanism of *Cisco IOS*.

IOS manages available free memory via a series of memory pools, which are essentially heaps in the generic sense; each pool is a collection of memory blocks that can be allocated and deallocated as needed. And the manager is memory pool manager, which is a very important component of the kernel [7]. The command, "show memory", lists memory pool details. And we can use a variation of this command, "show memory free", to display the free blocks in each pool. Obviously, it illustrates that blocks are connected together with each other and neighboring blocks are pointing at each other. That is to say, the entire heap of IOS is one big double-linked list [8], as shown at the left column in Fig. 1. Besides, every block has a header structure for management. Felix, a Cisco exploits

pioneer, gives a description of the organization for the memory block, and the header of process memory block is shown at the right column in Fig. 1. As can be seen, it contains, in addition to data, some relevant basic information such as the size of the memory block, two pointers to next and previous blocks of the memory, etc.

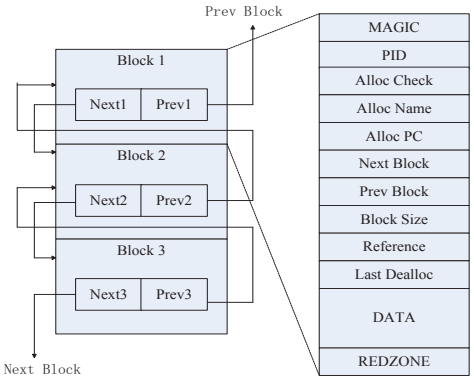


Fig. 1. Memory block of Cisco IOS

To maintain security and stability of the system, IOS uses "Check Heaps" process to check the integrity of their run code and storage stack structure. When there is an exception, the system will be forced to restart. For example, routers usually response some service requests without border checks, and if the request data exceeds the size of allocated storage space, the REDZONE field behind DATA will be overwritten. Then next time the "Check Heaps" process carried out to check abnormal memory, the system will be forced to restart.

D. ICMP Protocol Analysis

ICMP (Internet Control Message Protocol), a sub-protocol of TCP/IP protocol suite, is used to transmit control messages between IP hosts and routers. The control message refers to information of network itself, mainly including, whether the network is under barrier, if the host is reachable, whether the route is available or not and so on [9]. These control messages dont transmit user data, but play an important role in the transfer of user data. The specific protocol format is shown in Fig. 2 and the rest of Head has corresponding value according to different Type values. For example, when Type = 8 or 0, namely the usual Ping command, the third word of the packet is Identifier, and the forth word is Sequence Number. Data field can be defined by users.

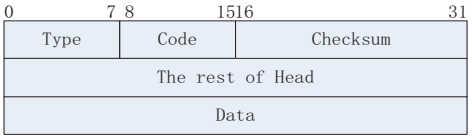


Fig. 2. ICMP protocol format

IV. Our Proposed Scheme

According to the description in the previous section, we design a vulnerability mining framework to detect Cisco router protocols. It completes a Fuzzer based on SFTCG model. The framework consists of four modules: tester module, target, monitor module and debugger module, as shown in Fig. 3.

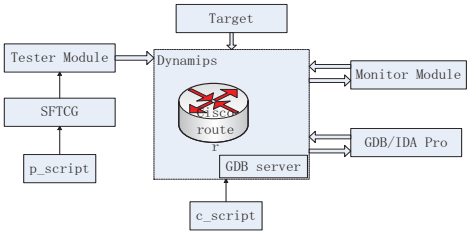


Fig. 3. Our vulnerability mining framework

1) Tester Module: The *p_script*, donating protocol script file, is a detailed description of the tested router protocol. It explains protocol type, corresponding port number, attributes in each field, constraints between them, vulnerabilities that may exist and so on. The Fuzzer parses it, and then conveys the result to SFTCG model to generate test-cases of the particular protocol, Tester Module using to test the Target router.

2) Test Target: Our Test Target is Cisco router. In this test, we use *Dynamips* [10], a hardware emulation program, to simulate Cisco routers. This solves the problem of obtaining the target hardware. Dynagen is the text-based front-end control system of Dynamips, which communicates with Dynamips in Hypervisor super-monitoring mode. As IOS run in Dynamips, in order to gain an insight of the inner working of Cisco IOS, we make a modification as in [11] to provide Dynamips the capability to interact with debuggers like GDB using GDB's standard debugging protocol. Besides, before testing, we also modify Dynamips, inserting some relevant modules, so that IOS can run better and interact with the test client by Dynagen more efficiently.

The *c_script file*, namely configuration file, is a description of the configuration of the entire virtual network, including parameters of the Cisco router, GDB remote debugging interface, the path of the system log files and so on. Before IOS executes, the Fuzzer parses the *c_script* file, configure Cisco router, and start its corresponding service.

3) Monitor Module: In this paper, we adopt three usual ways to monitor the status of the router [6]. First, enable SNMP service on the target router, and monitor its run status by sending Get/Set/GetNext/GetBulk Request of SNMP packet to port 161 of the router. Meanwhile, if the target router comes into a particular situation, such as down, it will send TRAP packet to monitor client. Second, check CPU utilization regularly, and you can confirm whether the router is attacked by DOS attacks. Finally, you can view the log file regularly created by dynamips to determine the status of the router.

4) Debugger: The debugger is implemented on the basis of Dynamips, implementing dynamic debugging IOS by using

TABLE I
SETUP FOR HOST & CISCO ROUTER

Host/Cisco router	Related parameters			
	Platform & Version	IP address	Tested protocol	GDB port
Host	Ubuntu 12.10	192.168.5.5	ICMP	12345
Router	C1700 12.4	192.168.5.1		
Capture Host	Win7	192.168.5.6		

GDB remote debugging technology [11]. And the IOS built-in GDB server serves as the debugging interface for Cisco researchers and developers [12]. Tester can debug IOS process dynamically by setting breakpoints to view status with debugger, such as GDB or IDA Pro. On the other hand, they can also view the data in memory and registers when exception comes out, such as target router crashes. All these result from the following assumption. If there is an invalid memory access exception or other abnormality, the debugger will stop just before the execution and immediately log the corresponding triggering test case, the registers and the code path, and then restart the router to test next test-cases.

V. IMPLEMENTATION AND EVALUATION

In order to verify the effectiveness of our developed Fuzzer tool, we perform an experiment on Cisco router. The specific implementation is as follows.

A. Setup

Cisco router is simulated by Dynamips, and Dynagen, as a front-end control system, provides control terminal a platform to interact with test target. According to the *c_script* file, we build a test LAN. In the LAN, local host serves as the tester and one of Cisco routers acts as the target, and their setup parameters are shown in Table I.

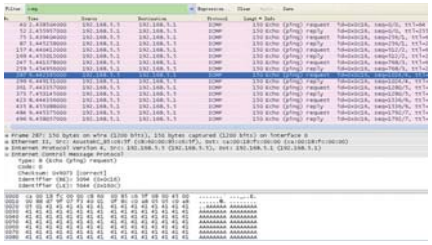
B. Evaluation

During testing process, the Fuzzer sent a large number of test-cases to target router, as shown in Fig. 4. Meanwhile, Monitor module keeps watch over the situation of network, especially the target, by sending SNMP package, checking log files and inspecting CPU utilization regularly. We also utilize Wireshark to capture the ICMP packets conveyed in the LAN. When an exception occurs, the debugger stops to view the data in memory and registers of IOS, and analyze and estimate the possibility of exploiting the certain vulnerability.

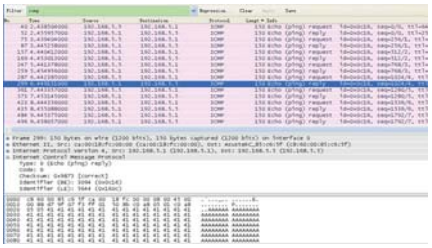
```
[09:28.47] initializing capture for test case #6700
[09:28.49] stopped PCAP thread, snagged 571 bytes of data
[09:28.51] initializing capture for test case #6701
[09:28.52] stopped PCAP thread, snagged 572 bytes of data
[09:28.54] initializing capture for test case #6702
[09:29.01] stopped PCAP thread, snagged 1082 bytes of data
[09:29.03] initializing capture for test case #6703
[09:29.11] stopped PCAP thread, snagged 1083 bytes of data
[09:29.14] initializing capture for test case #6704
[09:29.20] stopped PCAP thread, snagged 1514 bytes of data
[09:29.22] initializing capture for test case #6705
[09:29.22] stopped PCAP thread, snagged 1514 bytes of data
```

Fig. 4. Test-cases sent by Fuzzer

1) Test Result: Tester client can get the *echo_reply* packets, after sending plenty of *echo_request* packets in normal size in a short period of time. Then turn to another host and check the captured ICMP packets by Wireshark, shown in Fig. 5, and we find that the tested Cisco router responses to the *echo_request* from tester client. However, we find the CPU utilization of the router is abnormal, which has reached 98% to 99%, as shown in Fig. 6.



(a) Request phase



(b) Reply phase

Fig. 5. Result of sending plenty of ICMP packets

```
Router>show proc cpu
CPU utilization for five seconds: 99%/42%; one minute: 61%;
 PID Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min
   1           0         1          0  0.00%  0.00%  0.00%
   2           0        35          0  0.00%  0.00%  0.00%
   3         316        70       4514  0.72%  0.16%  0.03%
```

Fig. 6. CPU utilization of the tested Cisco router

When sending ICMP packets of about 64KB, we check the captured ICMP packets by Wireshark, shown in Fig. 7. It shows that there are only *echo_request* packets, but no *echo_reply* packets. That is to say, the target router refuses to response to the malformed request packets.

When there is an exception, the Cisco router in the modified Dynamips stops with the notes as in Fig. 8. Then we turn to GDB client of Debugger Model and see that the Cisco router is about to jump to the 0x41414141 memory address, which can be controlled by attacker [11] as shown in Fig. 9.

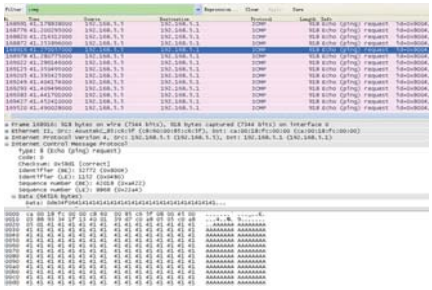


Fig. 7. The captured ICMP packets by Wireshark

```
---> ia_prev = 0x80A397A4
---> ia      = 0x41414140
```

Fig. 8. Dynamips notes

2) Result Analysis: After a deep study of the vulnerabilities discovered in the testing, we hold that only one malformed request packet cannot cause the server down, but a lot of ICMP packets in a short time interval or packets of almost 64KB, may lead the router to a situation, where IOS CPU utilization nearly 100% or it doesnt just respond directly on the request.

In the security testing on Cisco router, we detect the vulnerabilities of IOS to parse ICMP packets, including Ping of Death and Denial of Service. More importantly, the result validates the efficiency of Fuzzer based on *SFTCG*.

3) Solution: In order to avoid attacks on IOS of dealing with ICMP packets, some measures have been proposed. For example, the routers or hosts reject all ICMP packets in the network when the service is unnecessary. On the other hand, the router should limit ICMP packets bandwidth (or limit the number of ICMP packets), within a certain range. Also it can verify the size of the received ICMP packet, and discard them, if too large (greater than 64KB).

VI. Conclusion and Future work

In this paper, we have discovered vulnerabilities on router protocols implementing of Cisco IOS based on Fuzzing, and presented a novel Fuzzer based on SFTCG model, which excavates the vulnerability of parsing ICMP packets on Cisco router effectively. Meanwhile, the Fuzzer can be extended to be applied for other Cisco devices, such as Cisco switcher, to discover other protocols.

However, the proposed solution still has some drawbacks. For example, the Fuzzing testing process of monitoring is not automatic completely, for it needs some manual operations performed to cooperate with. Besides, there is an issue of combination explosion resulted from the multidimensional Fuzzing test-cases. In the next study, further work is needed to analyze IOS bug on other protocols, and possible extension of Fuzzer is essential to be applied to other routers to improve its own versatility. Of course, the way, to solve combination storm caused by multi-fuzzing, still requires further study.

```
(gdb) target remote :12345
Remote debugging using :12345
0x0001f0ff in ?? ()
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
0xa497a380 in ?? ()
(gdb) x/i $pc
=> 0xa497a380:      .long 0x0
(gdb) i r
r0             0x41414141       1094795585
r1             0x48ec7d82       1223458178
r2             0x482            1154
.
.
.
r25            0x41414141       1094795585
r26            0x41414141       1094795585
r27            0x41414141       1094795585
r28            0x41414141       1094795585
r29            0x41414141       1094795585
r30            0x41414141       1094795585
r31            0x41414141       1094795585
pc             0xa497a380       0xa497a380
msr            0x0  0
cnd            0x0  0
lr             0x41414141       0x41414141
cnt            0x0  0
xer            0x0  0
mq             0x0  0
(gdb) si
0x40414141 in ?? ()
```

Fig. 9. GDB client information

Acknowledgment

We sincerely thank all those contribute to our work and acknowledge Wireless Network Lab providing platform to experiment on our test.

References

[1] M. Sutton, A. Greene, and P. Amini, "Fuzzing: Brute Force Vulnerability Discovery," *Pearson Education*, 2007.

[2] D. Aitel, "The Advantages of Block-Based Protocol Analysis for Security Testing," *Immunity Inc.*, February, 2002.

[3] J. Xia, Ed., "Research for Buffer Overflow Vulnerabilities Based on Multi-Dimensional Fuzzing Technology," *Application Research of Computers*, vol. 28, no.9, September, 2011.

[4] M. Lynn, "The Holy Grail: Cisco IOS Shellcode and Exploitation Techniques," *Black Hat USA, Las Vegas, NV*, July 2005.

[5] A. Cui, J. Kataria, and S.J. Stolfo, "Killing the Myth of Cisco IOS Diversity," *USENIX Worshop on Offensive Technologies*, California, USA, 2011.

[6] Z. Wang, Y. Zhang, and Q. Liu, "A Research on Vulnerability Discovering for Router Protocols Based on Fuzzing," *IEEE International ICST Conference on Communications and Networking in China (CHINACOM)*, 2012.

[7] V. Bollapragada, C. Murphy, and R. White, "Inside Cisco IOS Software Architecture," *Cisco Press*, Indianapolis of USA, 2000.

[8] Z. Hu, Y. Zhu, Y. Ma, and Y. Han, "Cisco Internetwork Operating System Memory Management and Its Vulnerability Analysis," *Computer Engineering*, vol. 37, no. 20, October, 2011.

[9] ICMP. [Online]. Available: http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[10] Dynamips. [Online]. Available: http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator, Retrieved January 2011.

[11] Groundworks Technologies, Dynamips GDB Server mod project. [Online]. Available: http://www.groundworkstech.com/projects/dynamips-gdb-mod

[12] S. Muniz, and A. Ortega, "Fuzzing and Debugging Cisco IOS," *BlackHat Europe*, 2011.