TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.3, March 2014, pp. 1825 ~ 1831
DOI: http://dx.doi.org/10.11591/telkomnika.v12i3.4561

A Detection Method Based on Control Flow Graph for Cisco IOS Security

Liu Sheng-li, Gao Xiang, Zeng Cheng, Chen Li-gen
State Key Laboratory of Mathematical Engineering and Advanced Computing,
Zhengzhou 450002, China

Abstract

Aiming at the problem of current analysis and detection techniques against Cisco IOS security is not suitable for IOS integrality attack, this paper focuses on the Cisco IOS security detection techniques based on Control Flow Graph. First, the constructing method of Control Flow Graph is introduced. Then, a method to extract non-executed malicious code is proposed, which improves the effectiveness and accuracy of the analysis of malicious code. It provides support for rapid and effective detection of IOS integrality attack.

Keywords*: cisco IOS, control flow graph, integrality, security detection*

1. Introduction

With the growing demand for information, Computer networks and communication technology have obtained the fast development. The routers are an important part of the Internet and the basis of network interconnection, which directly affect the security of the entire network. In recent years, a large number of routers' hardware and software vulnerability is published, and lots of them had been directly exploited by attackers. The attackers could attack the network by it, and the existence of routers' vulnerability brings hidden troubles of the information security.

As the further research on Cisco IOS vulnerability, especially the IOS image integrity attack technique is proposed, there have been many research reports published about the IOS image integrity attacks [1, 2]. It greatly affects Cisco router safety. However, the existing security detection technologies of Cisco IOS are not suitable for such attacks. Currently, the analysis methods for Cisco IOS image code mainly include the static and dynamic analysis methods. The static analysis method is realized by IDA [1, 3] which is a static disassembly analysis tool. The advantage of the method is that the code is more comprehensive covered, and the interrelationship of function is clear. The disadvantage of the method is that the function analysis is not successful sometimes, a large number of strings are not associated, and the encryption code and disassembled code can't be successful to analysis. The dynamic analysis methods mainly include the analytical method for Cisco IOS based on gdb and the analytical method for Cisco IOS based on virtualization technology. The former is a method to analyze IOS by gdb debug module embedded in router in running state. The advantage of the method is to obtain real-time data. The disadvantage of the method is that the corresponding function of data can't be determined, and it's not useful for malicious code analysis. The latter is a means of building a virtual router platform that enables Cisco IOS can be loaded and be running on the platform, and the IOS debugging and analysis can be implemented during IOS system's normal operation. The advantage of the method is to dynamically trace and debug the running IOS system. The disadvantage of the method is that aiming at particular function or service of the IOS, it lacks of the adaptation to malicious code analysis of the IOS.

According to the above problems, in order to detect the integrity attack of IOS image better, based on router virtualization technology, this paper focuses on the Cisco IOS security detection techniques based on Control Flow Graph. First, the constructing method of Control Flow Graph is introduced. Then, a method to extract non-executed malicious code is proposed, which improves the effectiveness and accuracy of the malicious code analysis. It has good applicability for integrity attack of IOS image.

Received August 24, 2013; Revised September 21, 2013; Accepted October 10, 2013

2.Basic Concepts

Definition 1: Control Flow Graph [6, 7]. The control flow graph is a directed graph G=(V, E, ENTRY). In which, V is a set of nodes, and it's the basic block of procedure. E is a set of directed edges, and it represents the transfer relationship between basic blocks. ENTRY is the basic block of procedure entrance.

1. The basic block is composed of continuous performance instruction sequence, and has the following characteristics:

(1) The execution of basic block only begins with the first instruction of basic block, then according to instruction sequence in the basic block, once executed. Simultaneously, other basic blocks are not allowed to jump to a certain instruction of this block and be executed;

(2) Once the basic block is executed, it will not leave the block until all the instructions in the basic block are implemented. It will leave from the last instruction of block when the basic block is executed;

(3) Each basic block contains a branch instruction in the most, and the branch instruction is the last instruction in the basic block;

(4) Each node can have at most two immediate successors. For the node v with two immediate successors, its outgoing edges have attributes "T" or "F";

(5) For any node N, there exists a path from ENTRY to N.

2. E is the set of directed edges, which represents the relationship between the basic blocks. It has the following properties:

(1) The edge E_{ij} represents a possible control flow from v_i to v_j. v_i and v_j are the basic blocks, $v_i \in V$, $v_j \in V$, $E_{ij} \in E$;

(2) $e(v_i, v_j) \in E_{ij}$ represents the edge $v_i \to v_j$ in control flow graph. v_i is the immediate predecessor of v_j. v_j is immediate successor of v_i. If $\exists i, j(0 \le i \le j \le n)$, and $\exists E_{ij} \in E$, then v_i is called the precursor of v_j, v_j is called the successor of v_i. The set of all the precursors of node u is called precursor set of u, denoted $pre(u)$. The set of all the successors of node u is called successor set of u, denoted $tail(u)$.

(3) When $e(v_i, v_j) \in E_{ij}$, if v_j is the transfer destination of v_i when the transfer condition is not satisfied, then $e(v_i, v_j)$ is called the "F" edge of v_i. Otherwise, $e(v_i, v_j)$ is called the "T" edge of v_i. If there is no transfer instruction in v_i, then $e(v_i, v_j)$ is the "T" edge of v_i by default.

Definition 2: IOS image integrity attack techniques. The integrity of data refers to ensure that the information or data is not tampered with or the tampered data can be quickly found, in the process of transmission, information storage and data storage. The attack to IOS file integrity is a router attack method of using the vulnerability of IOS image integrity protection mechanisms, damaging the IOS file integrity and injecting malicious code, bypassing the memory protection mechanisms of IOS system when the malicious code running.

The attack is generally divided into six steps: ①Extract IOS image; ②Analysis of the original IOS image file; ③Construction of code; ④Code injection; ⑤Reconstruction of IOS file; ⑥IOS replacement.

3.Control Flow Graph Construction Method

3.1. Construction of Malicious Code Control Flow Graph

When capturing and filtering the execution instruction stream of malicious code, through statistics and analysis, we found that a portion of the malicious code may not be performed. In order to obtain a comprehensive understand about the execution flow of malicious code and provide more help for malicious code analysis, it needs to mine malicious code information which is non-executed. Here we use the method of building malicious code control flow graph.

The control flow graph of procedure can show the branches and paths in procedure, it can help to understand the internal structure of procedure. After capturing instruction stream

when the malicious code is running, through analysis of its branch instruction, the instruction stream sequence is divided into code blocks. The relationship between the blocks is analyzed based on jump relation of branch instruction. Thus the control flow graph for this part of code is generated.

To generate the control flow graph of malicious code, on the one hand, the call relationship and the dependency relationship between code blocks is clarified, and it can clearly reflect the execution flow of code. On the other hand, it's beneficial to understand the behavior of procedure with the data streaming information.

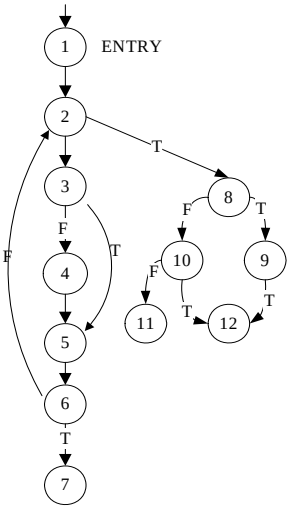

Figure 1. Control Flow Graph

The control flow graph of an independent section of instruction stream shown in Figure 1 is presented. In which, the node 1 is ENTRY of control flow graph. The node 2 is the direct precursor of node 3. The node 3 is the immediate successor of node 2. There is a path from node 1 to node 11, so that the node 1 is the predecessor of node 11, the node 11 is the successor of node 1. $e(2,8)$ is the "T" edge of node 2, and $e(2,3)$ is the "F" edge of node 2. Because of that there is no transfer instruction in the node 1, so $e(1,2)$ is the "T" edge of node 1 by default. For the node 10, its predecessor set and successor set are respectively as pre(10)={1, 2, 8} and tail(10)={11, 12}.

3.2. Extraction of Test Path

Using the method above, we can generate a control flow graph of procedure through an independent section of instruction stream, namely directed graph G=(V, E, ENTRY). Meanwhile, the nodes of having no immediate successor are possessed by following features: they are either a part of the malicious code that have been executed or the original performance function which is called by malicious code. To mine the non-executed portion of malicious code, we have to concern about the existence of those conditional branch instructions. Based on the directed graph G = (V, E, ENTRY), we should extract the nodes that have conditional branch instructions, setting up set N, and any $n \in N$. Setting up parameter sin(n), and it's used to represent the implementation situation of immediate successor of node n. If the edge "T" is implemented by node n, then sin(n)=1. If the edge "F" is implemented by node n, then sin(n)=2. Other types of node v set parameter sin (v) = 3. In Figure 1, the set of flow graph of procedure is N={2,3}, sin(2)=1, sin(3)=2.

4.The Discovery and Extraction Algorithm of Non-executed Code

To mine the non-executed portion of malicious code, it needs to begin with the node of set N. Different edges are performed though changing transmission conditions to achieve the goal of the extraction of non-executed code. If the malicious code implements to

node n, the transfer direction of node n is changed violently. On the one hand, some unreachable execution paths are extracted. On the other hand, maybe because of that the operating environment of process is changed compulsorily, it causes system collapse and impacts analysis to proceed. In order to make the discovery and extraction of non-executed code smoothly, so it needs less intervene the code running in this process. We can select the transfer direction through input information and the control of environmental resources. Therefore, we adopt the dynamical taint analysis method [8], and process data flow information based on taint analysis in fine-grained level, analyze and extract information of affecting nodal transfer direction. It can do good to help security analyzers, filter out interference instruction and extract valid instruction stream sequences [9].

In order to influence code running at least during the analysis process, in the process of discovery and extraction of non-executed code, we adopt the depth-first detection method. According to the theory of condition/decision coverage, based on the control flow graph of procedure, we propose a new discovery and extraction algorithm of non-executed malicious code. The algorithm is described as follows:

Input: The control flow graph information of procedure, the instructions and data sets that make each node in set N turn to the non-executed edge after the implementation.

Output: New control flow graph of procedure.

```
Begin:
do
{for each node n in set N// by node number
 if ((sin(n)==1))// if the edge “T” has been performed
 {Execute the edge “F” of node n;
   sin(n)= sin(n)+2;
   if (sin(n)==3) Remove the node n from the set N;
   if (the first executed instruction L of node n is in the interval Dirty)
{Build a new node m, implement the node m;
    sin(m)=0;
    Add the node m to set N;}
}
if ((sin(n)==2)|| (sin(n)==0)) // the edge “T” is not executed
{Execute the edge “T” of node n;
 sin(n)= sin(n)+1;
  if (sin(n)==3) Remove the node n from the set N;
 if(the first executed instruction L of node n is in the interval Dirty)
   {Build a new node m and implement the node m;
    sin(m)=0;
    Add the node m to set N;}
  }
}
```

In which, the interval Dirty gathers statistics and marks the non-executed malicious code memory range by comparison of instruction stream address range obtained and malicious code interval marked.

In the algorithm above, if L is not in the range of Dirty, usually there are several possibilities, needing manual processing:

1. L is not in the range of malicious code marked.At this time, L is beyond the analysis scope, there is no need to consider.

2. L is in the instruction sequence of existing direct graph G.At this time, the node of L needs to be segmented, and divided into two nodes with modifying relationships between nodes.

3. L is in the instruction sequence of other existing direct graphs.At this time, it needs analyzer to combine the direct graphs involved and modify the relationship between nodes involve in instruction L.

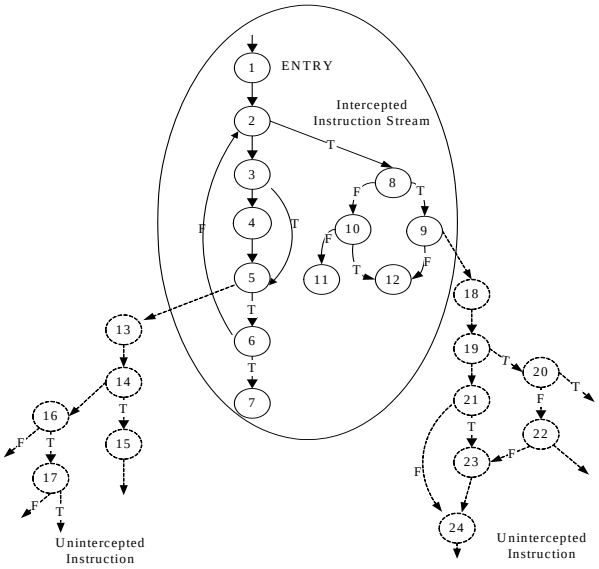

Figure 2. The code Coverage Situation Based on Control Flow Graph

Based on the contents shown in Figure 2, the algorithm is further explained as follows:

1. According to the sequence of node number, we input the trigger data obtained by data flow analysis method. It enters into the entrance of control flow graph of procedure, then it reaches the node 5. Because of sin(5)=1, the edge "T" of node has been executed, so the edge "F" of node 5 needs to be executed. After getting the first instruction L of node 5, we can judge out that L is in the interval Dirty and build a new node 13. Then the procedure continues to be executed until the complete implementation of node 13. At the same time, we assign sin(5)=sin(5)+2, and sin(5)=3, then remove the node 5 from the set N. At last, we add node m to set N, sin(13)=0, and N={9, 13}.

2. We input the data and make the procedure reach the next node 9. According to the process of step 1, the node 18 is added and the node 9 is removed. At the same time, we assign sin(18)=0 and N={13, 18}.

3. We input the data and make the procedure reach the next node 13. Because of sin(13)=0 and the edge "T" of node 13, we can get a new node 14. Then we assign sin(13)=sin(13)+1, sin(13) !=3 at this moment, the node 13 remains in the set N. At last, we assign sin(14)=0 and add the node 14 to set N, N={13, 14, 18}.

4. Continue the process until the set N is empty.

5. Experimental Verification

5.1. Experimental Environment

The environment topology is shown in Figure 3, the system of host A is WinXP, the Virtual machine software VMware is running on host A, it's used to build the virtual host C (Ubuntu9.04). This paper implements the IOS security detection system based on the discovery and extraction algorithm of non-executed code, and it is running on virtual host C. The network card of virtual host C connects with the physical network card of host A through bridge pattern. The virtual network interface of security detection system binds with the network card of host C.

This paper intended to attack the IOS image with the version number c2600-is-m. 122-8.T10.bin, and constructed the attack sample. The sample analyzed the function of hiding accounts, and the hiding account was "aaabb". When the router made user authentication, the attacker could bypass the normal authentication process to obtain administrator privilege by inputting the string "aaabb" into the system under IOS image integrity attack.

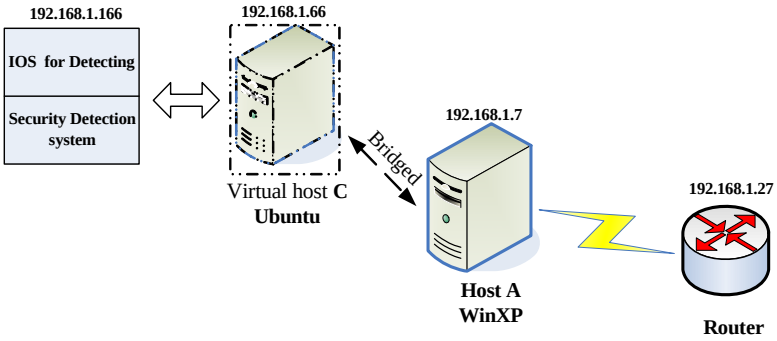

Figure 3. The Experimental Environment of Cisco IOS Security Detection System

5.2. Experimental Results

We constructed the control flow graph of procedure through the analysis of instruction stream and data flow, and analyzed the data combining with data flow information. The non-executed code was made extraction and identification, when the string sequence began with "a" was inputted in the user authentication procedure. In Figure 4, we obtained the new control flow graph through a period of automatic analysis. When the strings “b” and “abc” were inputted in turn, we could obtain the control flow graph that was shown in Figure 4(a). When the string “aaa” was inputted, we could obtain the control flow graph that was shown in Figure 4(b). List n ($n \in R$) represents basic block with the block number n. T:n($n \in R$) represents that the immediate successor of this block’s edge “T” is basic block n. F:n($n \in R$) represents that the immediate successor of this block’s edge “F” is basic block n. If some edge of block has no immediate successor or the immediate successor hasn’t been executed, then the corresponding value is 0.

It is not difficult to find that as long as there was a mistake, the procedure returned immediately when the input string were compared with the string“aaa” in recognizing user's identity process. When the string“aaa”was inputted by user, the malicious code made the operation of cyclic memory writing after the verification of the string. Then the malicious code executed other instructions after leaving the memory space range of malicious code through transfer instructions. We couldn’t make analysis on the function of follow-up executable code, because of that the follow-up instructions were not in the presupposition of memory space range of malicious code.

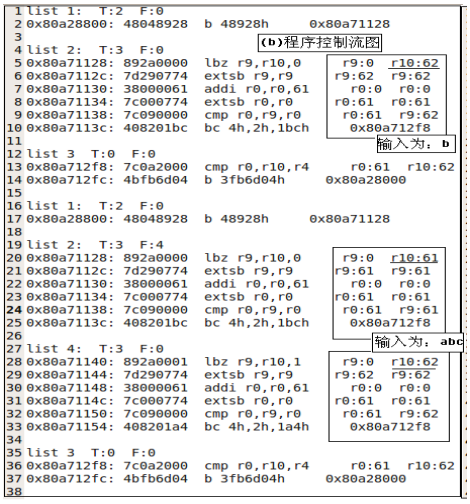

Figure 4(a). The Malicious Code Static Analysis

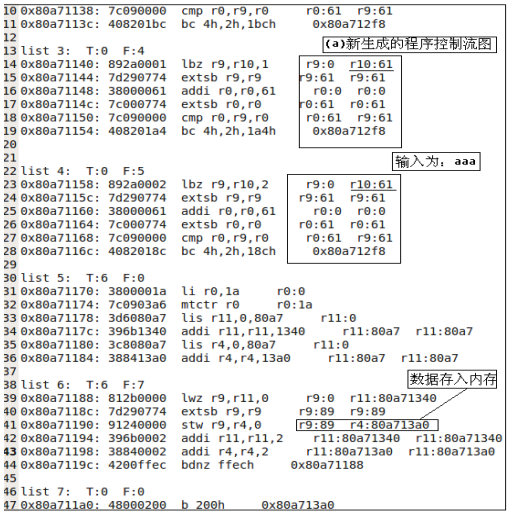

Figure 4(b). The Malicious Code Static Analysis

In addition, the tests of integrity attack for other IOS versions 11.3/12.0/12.1/12.3/12.4 were done by this paper. The experimental results show that the system can effectively analyze the malicious code, mine and extract the non-executed code.

6. Conclusion

This paper generates the control flow graph through the malicious code instruction stream and data stream information intercepted, mines the non-executed instructions of malicious code, proposes the discovery and extraction algorithm of non-executed code, and implements the Cisco IOS security detection system based on the algorithm. The experimental analysis demonstrates that the proposed algorithm was effective, the ability of Cisco IOS security detection system to analyze the malicious code of IOS integrity attack. It provides support for accurate, fast and effective detection of IOS integrity attack.

Acknowledgements

This paper is supported by National Natural Science Foundation of China (61309007), Zhengzhou Science and Technology Innovation Team Project (10CXTD150).

References

[1] YU Wang, WANG Haiyang, WANG Yadi, GU Xin. Study of Social Integrity Behavioral Mode Based On Multi-agent System. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(6): 3103-3108.
[2] Teddy Mantoro, Andri Zakariya. Securing E-mail Communication Using Hybrid Cryptosystem on Android-based Mobile Devices. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(4): 827-834.
[3] Sebastian 'topo' Muniz. Killing the myth of Cisco IOS rootkits: DIK (Da IOS rootKit). http://www.coresecurity.com/content/killing-the-myth-cisco-ios. 2008.
[4] Gyan Chawdhary, Varun Uppal, Cisco IOS ShellCodes/Backdoors. https:// www. blackhat.com/presentations/bh-usa-08/Chawdhary_Uppal/BH_US_08_Chawdhary. 2008.
[5] Sebastian 'topo' muniz, Ortega Alfredo.Fuzzing and Debugging Cisco IOS. https:// www.blackhat.com/bh-eu-11/MunizOrtega/BlackHat_EU_2011_MunizOrtega_Cisco_IOS-wp.pdf. 2011.
[6] MA Hong-tu, Zhao Rong-cai, Su Yan-bing. Analysis and Implementation of the Computation of Dominator in CFG. Computer *Science*. 2009; 36(3): 54-57.
[7] Holloway G, Smith D. The Machine-SUIF Control Flow Graph Library. http://www.eecs.harvard.edu/hube/software/nci/cfg.pdf, 2002.
[8] LI Jia-jing, Wang Tie-lei, Wei Tao, et al. A Polynomial Time Path-Sensitive Taint Analysis Method. *Chinese Journal of Computers.* 2009; 32(9): 1845-1855.
[9] YE Yong-hong, Wu Dong-ying, Chen Yang. Reverse platform based on fine-grained taint analysis. *Computer Engineering and Applications.* 2012; 48(28): 90-96.
[10] GONG Dun-wei, Zhang Yan. Novel Evolutionary Generation Approach to Test Data for Multiple Paths Coverage. *Acta Electronica Sinica*. 2010; 38(6): 1299-1305.