Taint Analysis of Single Instruction on Critical Infrastructure

Young Hyun Choi, Jung-Ho Eom[1] and Tai-Myoung Chung

Dept. Electrical and Computer Engineering, Sungkyunkwan University
Dept. Military Studies, Daejeon University
Dept. Information Communication Engineering, Sungkyunkwan University

yhchoi@imtl.skku.ac.kr, eomhun@gmail.com, tmchung@ece.skku.ac.kr

Abstract

Critical infrastructure is the backbone of our nation's economy and security. Cyber security and Communication works to prevent or minimize disruptions to critical information infrastructure in order to protect the public, the economy, and government services. And Software vulnerabilities can make possibility of a critical threat. There is lots of research being a large scale of software vulnerability. Critical infrastructure's security requirements are to process the large amounts of data in a short time. Taint analysis is becoming major technique in security analysis. In particular, proposed system focuses on tracing a taint analysis. Therefore, we proposed taint analysis method to determine by a single set of instruction and operands for threats detection. We proposed classification of binary level taint analysis that can find two types of threats, are control flow hijack and arbitrary memory write. Our evaluation results show that we can reduce number of instruction check.

Keywords: *Taint analysis, Binary analysis, Categorizing, Taint information*

1. Introduction

Critical infrastructure is the backbone of our nation's economy, security and health. Critical infrastructure is the assets, systems, and networks which are vital to the nation's security, economy and health [1]. Cyber security and Communications (CS&C) is responsible for enhancing the security, resilience, and reliability of the Nation's cyber and communications infrastructure. Cyber intrusions and attacks have increased dramatically over the last decade, exposing sensitive personal and business information, disrupting critical operations, and imposing high costs on the economy [2].

Our nation's critical infrastructures are highly interconnected and mutually dependent in complex ways, both physically and through a host of information and communications technologies [3]. Identifying, understanding, and analyzing such interdependencies are significant challenges. These challenges are greatly magnified by the breadth and complexity of our national critical infrastructures. These infrastructures, which affect all areas of daily life, include electric power, natural gas and petroleum production and distribution, telecommunications (information and communications), transportation, water supply, banking and finance, emergency and government services, agriculture, and other fundamental systems and services that are critical to the security, economic prosperity, and social well-being of the nation [3].

Taint analysis is major of technique for information flow. And taint analysis is available approach binary level without source code. Taint analysis works by tagging inputs to a

[1] Corresponding Author

program as tainted. And tainted data taints to other values that are influenced tainted inputs [4]. We can check for information well-known vulnerability in a application by marking inputs as tainted, and then checking whether they propagate to inappropriate outputs [5,6,7,8].

Malware propagates across computer systems leaking private information and harming usability. They use software vulnerabilities on the target system to do malicious activities [9]. We propose a method to reduce usage and access frequency of resources and times in taint analysis. We only track information of memories and registers that have higher possibility of being exploitable [4, 10]. Therefore we do not need to check instruction of unavailable threat. These characteristics lead to a faster analysis on critical infrastructure.

In this paper, we focus on found possibility of threats on the taint analysis binary level. Structure of this paper is as follows. In Section 2, we discuss current cyber security environment on critical infrastructure with other country's case, and classify the possibility of threats for this matter in Section 3. Section 4 evaluates our proposed taint analysis. Finally, the paper is concluded in Section 5.

2. Current Cyber security Environment on Critical Infrastructure

Critical infrastructure is a term used by governments to describe assets that are essential for the functioning of a society and economy. Most commonly associated with the term are facilities for [11]:

-electricity generation, transmission and distribution;

-gas production, transport and distribution;

-oil and oil products production, transport and distribution;

-telecommunication;

-water supply (drinking water, waste water/sewage, stemming of surface water);

-agriculture, food production and distribution;

-heating (*e.g.*, natural gas, fuel oil, district heating);

-public health (hospitals, ambulances);

-transportation systems (fuel supply, railway network, airports, harbors, inland shipping);

-financial services (banking, clearing);

-security services (police, military);

In USA, department of homeland security (DHS) is in charge of overall critical infrastructure. Cyber security on critical infrastructure is also part of the work of DHS. DHS plays a key role in securing the federal government's civilian cyber networks and helping to secure the broader cyber ecosystem through [2]:

- Partnerships with owners and operators of critical infrastructure such as financial systems, chemical plants, and water and electric utilities

- The release of actionable cyber alerts

- Investigations and arrests of cyber criminals, and

- Education about how the public can stay safe online.

Combating cyber threats is a shared responsibility. The public, private, non-profit sectors and every level of government have an important role to play.

An experienced team of professionals maintain cyber security cooperated with other organizations. And efficiently, each organization shared cyber security-related information with each other. DHS was able to effectively respond to cyber incidents, and provides technical support to owners and managers of critical infrastructure. And it spreads the notification of practical, appropriate vulnerability and potential security threats.

By leveraging the resources of the cyber crime center of ICE, DHS detects document fraud and identity together, and took part in the Internet survey of smuggling and financial fraud.

3. Taint Analysis

Taint analysis is a kind of data flow analysis. Resource data is influenced by tainted data on the data flows. Taint analysis is a data flow tracing mechanism which tracks incoming data from the user input throughout whole process. The reason of marking 'tainted (tagged)' is that data originating from the user input is untrusted. Operations on this data are kept tagging, and taint tagging propagates to the result of operations [4].

The purpose of taint analysis is to trace information flow between sources and sinks in accordance with the taint policies [12]. A taint policy exactly determines the way of tainted data flows as a program executing from influence of tainted value [5, 6, 7, 8].

Taint analysis is an analysis method which tracks arbitrary input data given to the program and examines how the input propagates as the program execution. Taint analysis has been applied to various fields, especially the field of computer security. There are a lot of researches being done on developing analysis frameworks to efficiently analyze programs [13, 14, 15]. Taint analysis can be divided into three main factors such as taint introduction, taint propagation and taint checking [9]. A detail on each factor is described in the following subsections [9].

3.1. Taint Description

We need initial input data to be set as a taint seed for taint analysis. Input data can be a network packet, interface input, files and so on. Predefined system’s register and shadow memory is untainted state before the start of taint analysis. When the input data enters the program, the memory or the register that saves the input data is marked (tagged) as tainted. Taint information is expressed in (value, taint) format. Every value computed by an instruction has its own taint value. When other instruction uses the previously computed value as an operand, taint information of its value should be accessed to propagate the taint information to other memory locations or registers.

3.2. Taint Propagation

From the initial input, taint information is propagated to other memories and registers after each execution of instructions in the program. Taint propagation rules define how the taint information is propagated from one place to others according to different instructions. We call these propagation rules as taint policy. Taint policy can differ according to the specific purpose of the taint analysis. In special cases, taint information can be sanitized when the result of some instruction is independent of whether its operands are tainted or not. For example, exclusive OR (XOR) on same register always yields 0, which is independent of the value in the register.

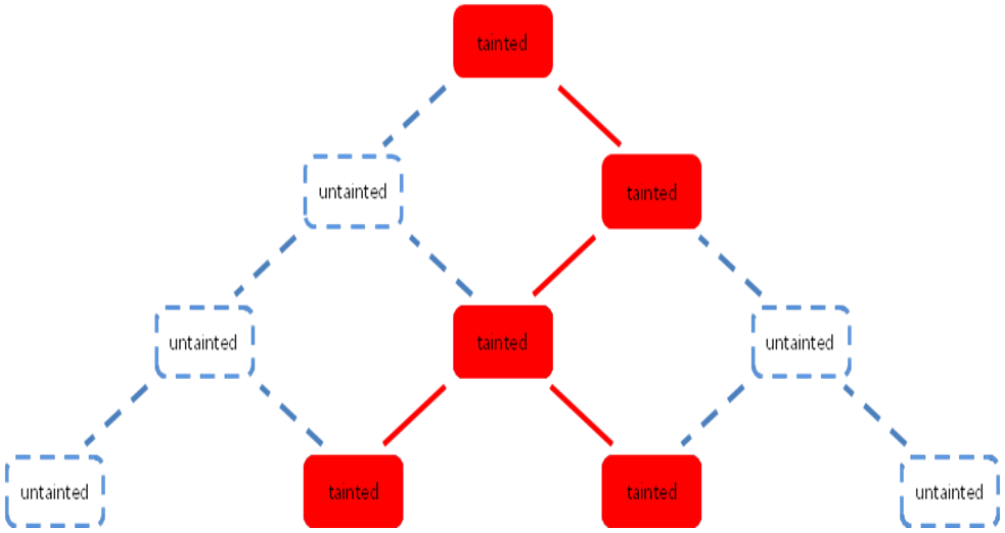

Figure 1. Taint Propagation

3.3. Taint Checking

Taint information can be used to determine the program state. For vulnerability detection, one might access the taint information of the program counter register to examine whether control flow of the program can be hijacked. In the case of malware analysis, we can taint sensitive data. We want to protect and examine if those data are sent to external servers.

4. Taint Analysis of Single Instruction

Taint analysis is a principal method for the exchange of trace information. And taint analysis is a method to be used for binary execution without source code. Taint analysis works to enter tags in the program, such as tainted. And tainted data taints to other values that are influenced tainted inputs.

We can be found in the well-known vulnerabilities with taint trace process. Taint analyzer traces at tainted input data, and then checks their propagation to influenced data. Also, Source code files are compiled into binary code. Structural features of source code level disappear what comes into assembly code [16].

We applied to the critical infrastructure to be determined threats by a single set of instruction and operands; we called ‘single instruction’. It is because security product of critical infrastructure requires fast velocity for processing the large amounts of data in a short time.

Taint analysis of single instruction can derive through threats such two types as control flow hijack and an arbitrary memory write.

Table 1. Taint propagation and available threat check classified by instructions

Instruction	Operand 1 (source)	Operand 2 (destination)	Propagation	Available threat
mov	T	T	Yes	arbitrary memory write
add	U	T	Yes	indeterminable
sub	U	T	Yes	indeterminable
inc	T	-	Yes	indeterminable
dec	T	-	Yes	indeterminable
and	A	A	Yes	indeterminable
or	A	A	Yes	indeterminable
xor	A	A	Yes	indeterminable
shift	T	C	Yes	indeterminable
jmp	T	-	Yes	control flow hijack
call	T	-	Yes	control flow hijack
cmp	A	A	Yes	control flow hijack

T: tainted, U: untainted, A: tainted or untainted, C: constant, -: not used operand

(state 'A': number of tainted resource is greater than 1)

We found possibility of threats within satisfying the conditions of taint propagation on the taint analysis binary level [4, 16].

If instruction 'mov' had both tainted operand 1 and tainted operand 2, then it has possibility of threat, called arbitrary memory write. Arbitrary memory write is threat that attacker can read arbitrary memory contents using vulnerability, next step is to write arbitrary data to arbitrary memory addresses. If instruction 'mov' had tainted operand 2, then resource of operand 1 derived from tainted data of operand 2; taint propagation.

Instruction 'add, sub, inc, dec, and, or, xor and shift' sets affect taint propagation, but are indeterminable fact that possibility of threat. If instruction 'add' had untainted operand 1 and tainted operand 2, then resource of operand 1 derived from tainted data of operation 2; taint propagation. If instruction 'sub' had untainted operand 1 and tainted operand 2, then resource of operand 1 derived from tainted data of operation 2; taint propagation.

If instruction 'inc' had tainted operand 1, then resource of operand 1 derived from itself; taint propagation. If instruction 'dec' had tainted operand 1, then resource of operand 1 derived from itself; taint propagation.

If instruction 'and' had tainted operand 1, untainted operand 2 and operand 2 are not zero, then resource of operand 1 derived from itself; taint propagation. If instruction 'and' had

untainted operand 1, tainted operand 2 and operand 1 are not zero, then resource of operand 1 derived from tainted data of operation 2; taint propagation. It is because instruction 'and' has characteristic that one or more operands are zero, and then result value is '0'.

If instruction 'or' had tainted operand 1, untainted operand 2 and operand 2 are not '1', and then resource of operand 1 derived from itself; taint propagation. If instruction 'or' had untainted operand 1, tainted operand 2 and operand 1 are not '1', and then resource of operand 1 derived from tainted data of operation 2; taint propagation. It is because instruction 'or' has characteristic that one or more operands are '1', then result value is '1'.

If instruction 'xor' had tainted operand 1 and untainted operand 2, then resource of operand 1 derived from itself; taint propagation. If instruction 'xor' had untainted operand 1 and tainted operand 2, then resource of operand 1 derived from tainted data of operation 2; taint propagation. Though, if operand 1 and operand 2 are the same resource (address), then result value is '0'.

If instruction 'shift' had tainted operand 1 and operand 2's constant value is less than operand 1's bit size, then resource of operand 1 derived from itself; taint propagation.

If instruction 'jmp, call' had tainted operand 1, then it has possibility of threat, called control flow hijack. And if instruction 'cmp' had at least one of tainted resource between operand 1 and 2, then it has possibility of threat, called control flow hijack.

Our proposed scheme efficiently focuses on finding vulnerabilities on critical infrastructure. Requirement of critical infrastructure's vulnerability checker is capable of testing at a faster as real-time processing. So, our proposed scheme is based on taint analysis because taint analysis technique traces tainted data flow. Taint policy of taint analysis checker sets check_only_tainted_inputdata that will checks tainted input data and taint propagation as is taint analysis checker. In addition, taint analysis checker examines our predefined single instruction of available threat. Our predefined single instruction of available threat is shown by Table 2.

Table 2. Available threat check classified by instructions

Instruction	Operand 1 (source)	Operand 2 (destination)	Available threat
mov	T	T	arbitrary memory write
jmp	T	-	control flow hijack
call	T	-	control flow hijack
cmp	T	U	control flow hijack
cmp	U	T	control flow hijack
cmp	T	T	control flow hijack

Our proposed taint analysis check policy had 6 cases of instruction and tainted operands. They consist an arbitrary memory write and 5 control flow hijacks. We decide on available threat with the single instruction.

First, arbitrary memory write case is instruction 'mov' had both tainted operand 1 and tainted operand 2 on single instruction. Arbitrary memory write is threat that attacker can read arbitrary memory contents using vulnerability, next step is to write arbitrary data to arbitrary memory addresses.

Next, a case of control flow hijack is attacker exploit vulnerabilities and gain control of the instruction. Control flow hijack threats consist a 'jmp', a 'call' and 3 'cmp's. Instruction 'jmp' had tainted operand 1 on single instruction. Instruction 'call' had tainted operand 1 on single instruction. Instruction 'cmp' had tainted operand 1 and untainted operand 2 on single instruction. Instruction 'cmp' had untainted operand 1 and tainted operand 2 on single instruction. Instruction 'cmp' had both tainted operand 1 and tainted operand 2 on single instructions.

5. Evaluation

In this section, we evaluate our proposal by tracing an example program and examining number of memories and registers that are available threats. Figure 2 is the source code of the program that we traced for the evaluation. It is a simple program that has two local character buffers to save user input strings. There is no boundary checking for the buffers, therefore it can cause a stack buffer overflow.

```
#include <stdio.h>
    int main(void)
    {
       char ID[100];
       char pw[100];
       char* pointer=NULL;
       gets(ID);
       gets(pw);
       printf("check certification");
       printf(ID);
       if(!strncmp(user,"admi",4) && !strncmp(pw, "pass",4))
       pointer = "Login SUCCESS";
       printf("Result: %s", pointer);
    }
```

Figure 2. Traced Program Code

Table 3. Summary of analysis results

Type	Number	Note
Total instructions	533134	Number of all instructions
Tainted instructions	6632	Number of tainted operand is not zero
Taint propagation instructions	5389	Number of taint propagation
Tainted flow instructions	1263	Two operands are tainted
Available threat instructions	1158	Available threat (arbitrary memory write+control flow hijack)
- Arbitrary memory write	317	mov instruction with tainted operand 1
- Control flow hijack	841	jmp, call instruction with tainted operand 1 cmp instruction with tainted operand

We inserted a long input string that triggers the overflow vulnerability in the program when trying to return to the caller of the main function and traced the execution. Over 533,134 instructions traced from the execution, 6632 instructions had at least one tainted operand. Tainted data influences to other instructions taint propagation; 5389 instructions. 1263 instructions had two tainted operands. And we find number of available threat instructions is 1158. 317 instructions had available threat of arbitrary memory write and 841 instructions had available threat of control flow hijack.

As a result, tainted instructions are 1.24 percentages of total instructions. Taint propagation instructions are 81.26 percentages of tainted instructions. Tainted flow instructions are 19.04 percentages of tainted instructions. Available threat instructions are 17.46 percentages of tainted instructions on proposed taint check policy. Arbitrary memory write is 27.37 percentages of available threat and control flow hijack is 72.63 percentages of available threat. Over 82 percentages of tainted instructions can be eliminated on proposed taint check policy of available threat check classified by instructions. High percentage of untainted instructions is due to long loop of instructions that manipulates the string. There are cases that need further consideration regarding untainted, for example byte operations, string operations etc.

6. Conclusions

Security for critical infrastructure is important resource to prevent a national disaster. Critical infrastructure's security requirements are to process the large amounts of data in a short time. Therefore, we proposed taint analysis method of to determine by a single set of instruction and operands for threats detection. We can find two types of threats such as control flow hijack and arbitrary memory write by binary level taint analysis. In the future works, we plan to design system that fine-grained classification technique using fewer resources and time.

We evaluated our proposal by tracing a test program. Test program is a simple program that has two local character buffers to save user input strings. There is no boundary checking for the buffers, therefore it can cause a stack buffer overflow. As a result, tainted instructions are 1.24 percentages of total instructions. Furthermore, over 82 percentages of tainted instructions can be eliminated on proposed taint check policy of available threat check classified by instructions.

References

[1] U.S. Department of homeland security, "What is Critical Infrastructure", USA.
[2] U.S. Department of homeland security, "Office of Cybersecurity and Communications", USA.
[3] http://en.wikipedia.org/wiki/Critical_infrastructure.
[4] Y. H. Choi and T. M. Chung, "A Framework for Dynamic Taint Analysis of Binary Executable File", ICISA 2013, **(2013)**, pp. 456-459.
[5] E. J. Schwartz, T. Avgerinos and D. Brumley, "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)", IEEE Symposium on Security and Privacy, **(2010)**.
[6] M. Kang, S. McCamant, P. Poosankam and D. Song, "DTA++: Dynamic taint analysis with targeted control-flow propagation", 18th Annual Network and Distributed System Security Symposium, **(2011)**.
[7] M. Scholten, "Taint Analysis in Practice", Vrije Universiteit Amsterdam, Amsterdam, **(2007)**, pp. 1-29.
[8] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software", Technical report, School of Computer Science Carnegie Mellon University, **(2004)**.
[9] J. W. Min, Y. H. Choi, J. H. Eom and T. M. Chung, "Eplicit Untainting to Reduce Shadow Memory Usage and Access Frequency in Taint Analysis", ICCSA2013, **(2013)**, pp.195-186.
[10] T. Avgerinos, S. K. Cha, B. L. T. Hao and D. Brumley, "AEG, Automatic Exploit Generation In", Proceedings of the Network and Distributed System Security Symposium, **(2011)**.
[11] S. M. Rinaldi, J. P. Peerenboom and T. K. Kelly, "Identifying, understanding, and analyzing critical infrastructure interdependencies", Control Systems, IEEE, vol. 21, no. 6**, (2001)**, pp.11-25.
[12] J. Caballero, N. M. Johnson, M. G. Kang, S. McCamant, P. Poosankam and D. Song, "Crash Analysis with Bitblaze", Blackhat, **(2010)**, USA.
[13] D. Brumley, I. Jager, T. Avgerinos and E. J. Schwartz, "BAP: A Binary Analysis Platform", CAV 2011, LNCS, vol. 6806, Springer, Heidelberg, **(2011)**, pp. 463–469.
[14] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam and P. Saxena, "BitBlaze: A New Approach to Computer Security via Binary Analysis", Information Systems Security, **(2008)**.
[15] J. Clause, W. Li and A. Orso, "Dytan: A Generic Dynamic Taint Analysis Framework", Proceedings of the 2007 International Symposium on Software Testing and Analysis, ACM, **(2007)**.
16] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual", vol. 2A, Instruction Set Reference, USA, **(2013)** June.

Authors

Young Hyun Choi received his B.S. degrees in Information Communication Engineering and M.S. degrees in Electrical and Computer Engineering from Sungkyunkwan University, Korea in 2008 and 2010, respectively. He is currently in the doctoral course of Electrical and Computer Engineering at Sungkyunkwan University, Korea. His research interests are System Security, Binary Analysis, Taint Analysis, Malware and Network Security.

Jung-Ho Eom (corresponding author) received his M.S. and Ph.D. degrees in Computer Engineering from Sungkyunkwan University, Suwon, Korea in 2003 and 2008, respectively. He is currently a professor of Military Studies at Daejeon University, Daejeon, Korea. His research interests are information security, cyber warfare and network security.

Tai-Myoung Chung received his first B.S. degree in Electrical Engineering from Yonsei University, Korea in 1981 and his second B.S. degree in Computer Science from University of Illinois, Chicago, USA in 1984. He received his M.S. degree in Computer Engineering from University of Illinois 1987 and his Ph.D. degree in Computer Engineering from Purdue University, W. Lafayette, USA in 1995. He is currently a professor of Information and Communications Engineering at Sungkyunkwan University, Suwon, Korea. He is now a vice-chair of the Working Party on Information Security & Privacy, OECD.