UNIVERSITI PUTRA MALAYSIA

RUNTIME INTEGRITY VALIDATION OF EXECUTABLE C BINARIES USING TRUSTED PLATFORM MODULE

TEH JIA YEW

FK 2017 24

RUNTIME INTEGRITY VALIDATION OF EXECUTABLE C BINARIES USING TRUSTED PLATFORM MODULE

By

TEH JIA YEW

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfillment of the Requirement for the Degree of Doctor of Philosophy

March 2017

DEDICATION

The author wish to dedicate this thesis

to his mother and sister.

This thesis is also dedicated to the authors'

grandparents (both paternal and maternal) and father who had

left the world before this thesis can be completed.

Not forgetting as well,

the furry guardians of the house gates whose

unwavering dedications spans more than 3 generations.

Being the first in both the *TEH* (paternal)

and *TAN* (maternal) clans to pursue a Ph.D. to date ,

the author further wish to dedicate

this thesis to the future generations

of both clans as a motivator

to follow the authors' footsteps.

'May the *Force* be with them.'

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfillment of the requirement for the Degree of Doctor of Philosophy

RUNTIME INTEGRITY VALIDATION OF EXECUTABLE C BINARIES USING TRUSTED PLATFORM MODULE

By

TEH JIA YEW

March 2017

Chairman : Khairulmizam bin Samsudin, PhD
Faculty : Engineering

Software developers working under pressure and tight deadlines frequently fail to implement secure programming practices during software development stages. Both constraints are one of the major contributing factors leading to the exploitation of software vulnerabilities for malicious intentions. Furthermore, commercially developed software, owing to intellectual property protection, do not provide the source code – finished products are only shipped in the form of executable binaries. Unfortunately, the exploitation of executable binaries with vulnerabilities, especially those coded with the C programming language leads to malicious, undesirable effects such as buffer overflows, privilege escalation (spawning an attackers' shell) or premature termination of an executing binary (the C language possesses powerful memory manipulation features i.e. pointers). The following gaps were identified in these three related past efforts into the integrity validation of executables: CBones, Dytan and RTC. First, in CBones, these limitations were identified: a memory debugging tool called Valgrind is required for validation, does not mitigate the vulnerable executable and reports high overheads (Normalized Performance of 0.87). Second, in Dytan, these limitations were identified: requires the use of a dynamic binary instrumentation tool called PINTOOLS for validation, does not mitigate the vulnerable executable and reports high overheads (Normalized Performance of 0.85). Lastly, in RTC, these limitations were identified the executable source code is required for validation and heavy use of static analysis leading to high overheads (Normalized Performance of 1.10). In this thesis, we propose our framework for the runtime integrity validation of executable binaries: Runtime Integrity Validation of Executable Binaries or RIBS. Our framework merges both static (for offline profiling) and dynamic (for runtime validation) analysis techniques for the runtime validation of the integrity of executables compiled with the *C* programming language. The integrity validation metadata of a trusted, origin executable is stored in a Trusted Platform Module (TPM) hardware register to prevent tampering. Such ensures that the executable binary integrity validation metadata can be totally trusted. In the security evaluation of RIBS, we had subjected RIBS to mitigate 10 categories of buffer overflow attack patterns in the Wilander and Kamkar testsuite

(encompassing the stack, heap and data/bss executable userspace memory areas), 5 real world shellcodes and 3 real world applications with buffer overflow vulnerability. RIBS is successful in the detection and termination of *all* 18 attacks patterns deployed. In terms of performance evaluations, overhead was measured in terms of CPU execution time [via GNU clock()]. We measured the CPU execution time of RIBS and compared the results with the CPU execution time of two major categories of attack mitigation mechanism deployed in the testbed Fedora Core 20 Linux OS: Address Space Layout Randomization (ALSR) and 5 other attack mitigation mechanisms implemented via the *gcc* compiler. Performance evaluations reveal that RIBS reported highest Normalized Performance (NP) of 0.68, which is the lowest as compared to CBones (0.87), Dytan (0.85) and RTC (1.10). Conclusively, RIBS performs marginally better as compared to all three efforts (CBones, Dytan and RTC) which requires the use of memory debugging tools for integrity validation of executable binaries. RIBS does not require the use of any tools. RIBS is able to detect integrity violations caused by these categories of violations: all forms of buffer overflow attacks mounted via the Wilander and Kamkar testsuite and real world privilege escalation attack shellcodes. Furthermore, as a last line of defence, RIBS is able mitigate integrity violations in executables via the runtime termination of the offending executable. This feature not available in CBones, Dytan and RTC.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk Ijazah Doktor Falsafah

PENGESAHAN KESAHIHAN PROGRAM C SEWAKTU PERLAKSANAAN DENGAN MENGGUNAKAN MODUL PLATFORM BOLEHPERCAYA

Oleh

TEH JIA YEW

Mac 2017

Pengerusi : Khairulmizam bin Samsudin, PhD
Fakulti : Kejuruteraan

Jurutera perisian sering bekerja di dalam situasi di mana masa yang diberikan untuk menyiapkan produk perisian adalah amat terhad, Maka, jurutera perisian menghadapi tekanan yang besar. Akibatnya, amalan pengaturcaraan yang selamat tidak dipraktikkan dengan sempurna ketika perisian dibangunkan. Ini merupakan faktor yang menyumbang kepada eksploitasi kelemahan di dalam perisian untuk tujuan berniat tidak murni. Tambahan lagi, perisian komersil, disebabkan oleh keperluan untuk melindungi hak cipta, tidak memberikan kod sumber – produk yang telah siap dipasarkan dalam bentuk binari yang boleh digunakan/dilancarkan ataupun dikenali sebagai *executable binaries*. Malangnya, eksploitasi kelemahan di dalam *executable binaries,* terutamanya yang dikodkan menggunakan bahasa pengaturcaraan *C*, mengakibatkan kesan-kesan yang tidak diingini, contohnya *buffer overflow*, *privilege escalation* (salah satu kesan ialah penggodam dapat menguasai terminal ataupun *shell* mangsa) mahupun menamatkan perlaksanaan perisian yang sedang digunakan ataupun lebih dikenali dengan penamatan pra-matang (bahasa pengaturcaraan *C* mempunyai kebolehan memanipulasi memori computer yang canggih). Kelemahan dikesan di dalam tiga usaha masa lampau yang bertujuan untuk mengesahkan kesahihan program: *CBone*s, Dytan dan RTC. Pertama, kelemahan - kelemahan berikut dikesan di dalam CBones: pengesahan kesahihan program memerlukan penggunaan peralatan perisian Valgrind, tiada langkah susulan penyelesaian terhadap program yang tidak sahih dan overhead yang tinggi dilaporkan (*Normalized Performance* bernilai 0.87). Kedua, kelemahan-kelemahan berikut dikesan di dalam Dytan: pengesahan kesahihan program memerlukan penggunaan peralatan perisian *Pintools* yang berupaya melakukan instrumentasi program dinamik, tiada langkah susulan penyelesaian terhadap program yang tidak sahih dan overhead yang tinggi dilaporkan (*Normalized Performance* bernilai 0.85). Akhir sekali, RTC mempamerkan kelemahan – kelemahan yang berikut: kod sumber program diperlukan bagi tujuan pengesahan dan penggunaan analisis statik yang melarat menyumbang kepada overhead yang tinggi (*Normalized Performance* bernilai 1.10) .Maka, di dalam tesis ini, kami membentangkan cadangan penyelesaian (*framework*) untuk mengesahkan integriti (ataupun kesahihan) perisian

di dalam bentuk *executable binaries*, yang dinamakan *Runtime Integrity Validation of Executable Binaries* ataupun RIBS. Pengesahan kesahihan dilakukan di waktu perlaksanaan program. Cadangan penyelesaian kami menggunakan kombinasi teknik analisis statik (untuk pengumpulan maklumat di luar talian ataupun *offline*) dan dinamik (untuk penentuan kesahihan program sewaktu perlaksanaan). *Metadata* yang digunakan bagi tujuan pengesahan integriti perisian asal disimpan di dalam daftar perkakasan *Trusted Platform Module*. Ini bertujuan mengelakkan sebarang gangguan terhadap *metadata* jesterunya memastikan kebolehpercayaan *metadata* tersebut. Penilaian keberkesanan dilakukan ke atas RIBS melibatkan yang berikut: 10 jenis serangan *buffer overflow* sepertimana yang dikelaskan di dalam ujian *Wilander and Kamkar* , 5 jenis serangan shellcode dan 3 jenis serangan aplikasi sebenar yang mempunyai kelemahan *buffer overflow* .Keputusan penilaian keberkesanan ke atas RIBS mendapati bahawa RIBS berjaya mengesan dan menamatkan kesemua 18 jenis serangan yang dilancarkan ke atas RIBS. Penilaian prestasi ke atas RIBS dilakukan dengan mengukur *overhead* hasil masa perlaksanaan CPU. Ini dilakukan dengan menggunakan fungsi *GNU* clock(). Masa perlaksanaan CPU untuk RIBS diukur dan dibandingkan dengan masa perlaksanaan CPU untuk dua jenis kaedah penangkis serangan perisian yang ditemui di dalam sistem pengoperasian *Fedora Core 20*, iaitu *Address Space Layout Randomization* (ASLR) dan 5 kategori penangkis serangan perisian yang digabungkan ke dalam *gcc compiler*. Sistem pengoperasian ini digunakan sebagai *testbed* untuk RIBS. Keputusan penilaian prestasi melaporkan *Normalized Performance* (NP) paling tinggi dengan nilai 0.68. Nilai 0.68 merupakan nilai yang terendah jika dibandingkan dengan CBones (0.87), *Dytani* (0.85) dan RTC (1.10). Kesimpulannya, RIBS mencatatkan persembahan yang lebih baik berbanding dengan CBones, Dytan dan RTC. Ketiga - tiga usaha yang dinyatakan memerlukan penggunaan perisian pemeriksaan ingatan komputer untuk berfungsi. RIBS tidak memerlukan penggunaaan sebarang perisian untuk tujuan pengesahan kesahihan program. RIBS berupaya mengesan semua kategori serangan *buffer overflow* yang terkandung di dalam pakej ujian Wilander and Kamkar dan juga berupaya mengesan serangan jenis shellcodes. Tambahan lagi, RIBS berupaya menamatkan perlaksaan program yang disahkan tidak sahih. Ini merupakn satu langkah pencegahan yang tidak dilengkapi di dalam CBones, Dytan and RTC.

ACKNOWLEDGEMENTS

Faithful thanks and deepest appreciation are extended first to my main supervisor, Dr. Khairulmizam bin Samsudin; and not forgetting my co supervisors: Associate Professor Dr. Nur Izura binti Udzir and Associate Professor Dr. Shaiful Jahari bin Hashim - for their confidence placed on me, patience, unrelenting trust, charismatic guidance, prevailing assistance in all aspects, valuable suggestions, comments and advice; from the beginning of this thesis and till the curtains were drawn. I am also indebted to the Malaysian Ministry of Higher Education (MOHE) for the award of the myBrain 15 (myPhD) scholarship in support of my Ph. D. candidacy in UPM.

I started my doctoral studies 8 years ago, back in 2009 with limited knowledge on computer systems security (my first and Masters degrees were in *Physics*), presenting much mundane and half baked (sometimes even quarterly baked) research proposals and ideas. My supervisors took on my half baked ideas with full patience, professionalism, enthusiasm and consistently suggested numerous fine improvements, without which this thesis would not have been possibly being completed. Credit is also given to anyone who had either directly or indirectly contributed to the completion of this thesis and also this research project.

I certify that a Thesis Examination Committee has met on 1 March 2017 to conduct the final examination of Teh Jia Yew on his thesis entitled "Runtime Integrity Validation of Executable C Binaries using Trusted Platform Module" in accordance with the Universities and University Colleges Act 1971 and the Constitution of the Universiti Putra Malaysia [P.U.(A) 106] 15 March 1998. The Committee recommends that the student be awarded the Doctor of Philosophy.

Members of the Thesis Examination Committee were as follows:

Sharifah Mumtazah Syed Ahmad Abdul Rahman, PhD
Associate Professor
Faculty of Engineering
Universiti Putra Malaysia
(Chairman)

Abd. Rahman bin Ramli, PhD
Associate Professor
Faculty of Engineering
Universiti Putra Malaysia
(Internal Examiner)

Azizol bin Hj. Abdullah, PhD
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

Fu-Hau Hsu, PhD
Professor
National Central University
Taiwan
(External Examiner)

NOR AINI AB. SHUKOR, PhD
Professor and Deputy Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 28 April 2017

This thesis was submitted to the Senate of the Universiti Putra Malaysia and has been accepted as fulfillment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Khairulmizam bin Samsudin, PhD
Senior Lecturer
Faculty of Engineering
Universiti Putra Malaysia
(Chairman)

Nur Izura binti Udzir, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Shaiful Jahari bin Hashim , PhD
Associate Professor
Faculty of Engineering
Universiti Putra Malaysia
(Member)

____

ROBIAH BINTI YUNUS, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date

Declaration by graduate student

I hereby confirm that:

- this thesis is my original work;
- quotations, illustrations and citations have been duly referenced;
- this thesis has not been submitted previously or concurrently for any other degree at any institutions;
- intellectual property from the thesis and copyright of thesis are fully-owned by Universiti Putra Malaysia, as according to the Universiti Putra Malaysia (Research) Rules 2012;
- written permission must be obtained from supervisor and the office of Deputy Vice-Chancellor (Research and innovation) before thesis is published (in the form of written, printed or in electronic form) including books, journals, modules, proceedings, popular writings, seminar papers, manuscripts, posters, reports, lecture notes, learning modules or any other materials as stated in the Universiti Putra Malaysia (Research) Rules 2012;
- there is no plagiarism or data falsification/fabrication in the thesis, and scholarly integrity is upheld as according to the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) and the Universiti Putra Malaysia (Research) Rules 2012. The thesis has undergone plagiarism detection software

Signature: ________________________________ Date: __________________

Name and Matric No.: Teh Jia Yew , GS24269

Declaration by Members of Supervisory Committee

This is to confirm that:

- the research conducted and the writing of this thesis was under our supervision;
- supervision responsibilities as stated in the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) were adhered to.

Jabatan Kejuruteraan Sistem Komputer dan Komunikasi
Fakulti Kejuruteraan
Universiti Putra Malaysia
43400 UPM Serdang
Selangor Darul Ehsan

Signature: ____________________

Name of Chairman of Supervisory Committee: Dr. Khairulmizam bin Samsudin

ASSOC. PROF. DR. NUR IZURA UDZIR
Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor

Signature: ____________________

Name of Member of Supervisory Committee: Associate Professor Dr. Nur Izura binti Udzir

Signature: ____________________

Name of Member of Supervisory Committee: Associate Professor Dr. Shaiful Jahari bin Hashim

Jabatan Kejuruteraan Sistem Komputer dan Komunikasi
Fakulti Kejuruteraan
Universiti Putra Malaysia
43400 UPM Serdang
Selangor Darul Ehsan

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE LISTINGS

LIST OF ABBREVIATIONS

COT	Chain of Trust
EK	Endorsement Key
glibc	GNU C Library
GRUB	Grand Unified Bootloader
IAE	Integrity Assessment Engine
IMA	Integrity Measurement Architecture
IVM	Integrity Validation Mechanism
IP	Instruction Pointer
OS	Operating System
PCE	PCR Extender
PCR	Platform Configuration Register
ROT	Root of Trust
SAP	Static Attestation Protocol
TCG	Trusted Computing Group
TDDL	Trusted Device Driver Layer
TPM	Trusted Platform Module
TSPI	TCG Service Provider Interface
TSS	Trusted Software Stack
VIM	Vulnerability Identification Mechanism
VMM	Vulnerability Mitigation Mechanism

CHAPTER 1

INTRODUCTION

1.1 Introduction

To date, major security reporting sites for example, Packetstorm Security (Packetstormsecurity, 2016) and US CERT National Vulnerability Database (NIST, 2016) continues to report software vulnerability disclosures with no lesser than 5000 cases from the year 2006 to 2015. The highest number of cases was reported in 2014, with an estimated 8000 cases. Software vulnerabilities also plague open source applications and OSes (e.g. Debian and Red Hat), touted for its stability and security due to the availability of source codes for validation.

Globally reputable consultancy firm Pricewater House Cooper (PWC) reported in a recent 2015 Global State of Information Security (IS) Survey 2015 that IS breaches worldwide organizations faced a 92% increase (or USD $ 20 million) compared to 2013 (Consultants, 2016). Security breaches occur in numerous forms, one common and direct method of security breach is via the deployment of hijacked software applications – in which the integrity of either the source code or the binaries had been compromised.

One example of such breach occurs in Juniper's firewall and router operating systems (OS). A third party (hackers and even law enforcement – the U.S. National Security Agency) was able to stick unauthorized codes into the OS for malicious intentions – such as espionage – for information gathering intentions (Bort, 2015).

One factor contributing to the cause of integrity violations in software –coding errors occurring over products developed over a tight deadline and lack of sufficient quality control testing prior to deployment or release. Furthermore, even in the event if bugs or errors were discovered, mitigation attempts were often too time consuming or lacked of success – such is attributed to the unavailability of the software's source code.

Even if the source code is available, time would be consumed to comprehend the intricacies of the code to unravel technical details pertaining to the code e.g. architecture or functionality. We therefore stress that, there is strong motivation for runtime integrity validation of code binaries.

1.2 Problem Statement

The first major obstacle towards the integrity validation of software lies in the unavailability of source code. Such is a common practice in commercially developed software, whereby only the executable binary is provided to the end user. The weakness or research gap demonstrated by the efforts listed below is that , in the absence of source code, integrity validation of software faces challenges in terms of the implementation of a few well established methods , for example (Lozano et al., 2015), REDAS (Kil et al., 2009) , AutoPAG (Lin et al., 2007) , Daikon (Ernst et al., 2007) and RTC (Yong, 2004). All require the source code for the implementation of the integrity validation mechanism in their respective proposed solutions.

Second, in some proposed hardware based solutions towards integrity validation of software, increased overheads were incurred at the implementation stage due to the required support of some form of hardware which is not found ubiquitously or requires some form of modifications either and the design or at the instruction level (Tuck et al., 2004) and (Ozdoganoglu et al., 2006).

The research gap or weakness demonstrated by (Tuck et al., 2004) is that the proposed solution is emulated (inside Bochs emulator) and not implemented in real time systems. Hence, the performance results reported are valid only for the sandboxed Bochs emulator.

On the other hand, the research gap demonstrated by SmashGuard (Ozdoganoglu et al., 2006) lies in the requirement of a hardware stack for protecting return address and the protection offered by SmashGuard does not extend to non stack areas in the memory, namely the heap, data and bss regions.

TPM solves both the problems of excessively high execution overheads and that of the need for specific hardware. The TPM is a low cost component (approximately USD 20) equipped with security features permitting user or metadata authentication and attestation. The TPM chip has, since its introduction in 2002, has been included on motherboards of both high and low end computers, hence the TPM chip presence is now ubiquitous (Group, 2008).

Furthermore, dynamic analysis solutions on integrity validation work via dynamically performing runtime checks on running binaries. This method work upon binary execution via dynamically checking for known traces of vulnerabilities – for example placement of stack canaries in the gcc code compiler or memory access only in permitted areas, as per (Cowan et al., 1998) ,(Zhou et al., 2004) and (Qin et al., 2006) . Apart from hardware based solutions, this method also contributes to the increased overhead at during execution. (Zhou, Liu, et al. (2004)

Essentially, training is required as a form of input in heuristics based solutions, especially prior to deployment. Heuristics based solutions incorporate statistical methods, which utilizes signatures for diagnosis (functions similar to antivirus). Some work in this category are as per (X. Chen et al., 2015), (Brumley et al., 2007), (H. J. Wang et al., 2004) and (Singh et al., 2004).

Vulnerabilities diagnosis is achieved via deviations from program executions obtained via deployment of statistical algorithms or methods. The major research gap or drawback of this technique is that despite being automated, both applied statistical mitigation methods or algorithms require time consuming input training prior to deployment. Another drawback lies in whether both methods are capable of gathering inputs from all forms of vulnerabilities – such exposes both techniques to false negatives.

Third, lies the question of how do someone trust the results of integrity validation, i.e. how can the executable binary integrity metadata (or guard) be trusted? Proposed validation mechanism can operate effectively but the software integrity cannot be totally guaranteed unless the executable binary integrity metadata can be attested as trusted.

The major research gap or weakness of current and past efforts into integrity validation of executables lies in the failure to guarantee the integrity of the validation metadata itself. Some examples are as follows: the use of shadow stack (Dang et al., 2015) or return address stored in shadow memory as the integrity validation metadata for executables, as per (Serebryany et al., 2012) , (Nethercote et al., 2007) and (Y. J. Park et al., 2004).

Limited work utilizes the TPM in the integrity validation of executable binaries or metadata, despite being introduced in 2002: work (Sailer et al., 2015), (Kil et al., 2007), (Gu et al., 2008) and (Gu et al., 2010).

Fourth, efforts into the evaluation of the integrity of executables focused only on the evaluation process itself while neglecting the need to perform mitigation as a follow up action on executables that fail the integrity evaluation process.

Fifth, related efforts into the integrity validation of executables lacked effectiveness evaluations conducted using comprehensive attack testsuite such as RIPE (Wilander et al., 2011), which covers all known types of buffer overflow attacks. Buffer overflow attack is the most common type of vulnerability found in software.

Hence, the major research gap or weakness of these efforts: HCFI (Christoulakis et al., 2016), Shadow Stack (Dang et al., 2015), Isomeron (Davi et al., 2015), Per Input CFI (Niu et al., 2015), PHUKO (Tian et al., 2014), CCFIR (C. Zhang et al., 2013), (Gu et al., 2010), CBones (Kil et al., 2007), AccMon (Zhou, Liu, et al., 2004) and iWatche*r* (Zhou, Qin, et al., 2004) shows that the focus is only on integrity evaluation (i.e. detection) but neglected on mitigation (i.e. prevention). Effectiveness evaluations were not conducted using comprehensive attack testsuite such as RIPE, hence some form of vulnerability may not be detected and mitigated.

1.3 Objectives

In an attempt to solve the problems outlined above, we propose in this thesis, our framework for achieving runtime software integrity in the absence of the source code – whereby only the binary is available. Our framework merges technique from both static and dynamic analysis – i.e. more conveniently termed as hybrid analysis.

The aim of this thesis is to develop a trusted framework to evaluate the integrity of executable binaries without the availability of source code. This thesis thus has the following objectives towards achieving the aim and solving the problem statements outlined in the previous section:

a. to undertake work for the development of the modules or mechanism constituting the proposed framework and to evaluate the proposed framework in terms of effectiveness and performance.
b. to protect executable binary validation metadata using Trusted Platform Module (TPM). The TPM prevents the tampering of such metadata, as such the proposed framework can be trusted and would never lie on the integrity of the validation metadata. Hence, our proposed framework introduces a mechanism (i.e. the Vulnerability Identification Mechanism or VIM) which functions towards ensuring that the integrity of the executable binary validation metadata can be guaranteed via the use of TPM.
c. to develop a mitigation mechanism to counter any threats posed by executables that failed the integrity evaluation by our proposed framework.

1.4 Scope and Assumptions

In terms of scope, our framework can be applied for the runtime integrity validation of various compiled programming languages such as C, C++ and Java. In this dissertation, focus is on the runtime integrity validation of executables coded with the C programming language. The justification for this choice is that C possesses powerful memory manipulation features – pointers being a good example. Hence, integrity violations in executable C binaries lead to effects of integrity violations such as buffer overflow and privilege escalation attacks.

Our proposed framework shall attempt to identify and capture integrity violations causing buffer overflows and in executable C binaries. The executables are compiled using the gcc compiler, while we note that there exits numerous types of C compilers e.g. Intel, Clang, Micro C compiler, C-Parser etc. Furthermore, different compilers are available for various OS platforms e.g. Windows, Unix, OS/2 or Mac OS. We propose that evaluation using alternative compilers and in alternative OSes, as future work.

We further assess the effectiveness of security mechanisms in our testbed platform – Fedora Core Linux version 20 (FC 20) - in mitigating two categories of integrity violation in software and binaries: buffer-overflow attacks and privilege escalation attacks.

Effectiveness evaluations were performed using the comprehensive RIPE testsuite (Wilander et al., 2011), which encompasses all known forms of buffer overflow attacks. The rules used in our proposed framework for the integrity validation of executables were derived from CBones (Kil et al., 2007). Performance evaluation shall be carried out via measuring the CPU Time of an executable at runtime. The GNU clock() tool is used for obtaining the CPU Time (GNU, 2015a).

1.5 Motivation for our proposed framework

The motivation behind our approach is as follows:

a. This thesis serves to complement the existing work on leveraging TPM for the integrity validation of executables in the absence of the source code.

b. while credit must be given to majority of the work into integrity validation of software for effectiveness test utilizing real world buffer overflow exploits, we found that none of the literature presented in Chapter 2 conducted effectiveness tests not even with the most short and simple but lethal shellcodes capable of privilege escalation attacks.

   We are of opinion that an additional work on effectiveness of software integrity validation solution tested with shellcodes would complement the wide array of effectiveness tests conducted on real world exploits. In this thesis, we propose to subject our framework to evaluate real world shellcodes obtained from the wild, in order to gauge the effectiveness of our framework in terms of security and performance.

a) With the exception of all literature listed in Section 2.4 of Chapter 2, majority of the literature mentioned in Chapter 2 neglected addressing the issue of trust in the integrity validation metadata (Chiueh, 2001). All assume that the integrity validation metadata can be totally trusted.

   Questions arise on the reliability of the metadata itself should the metadata be compromised. Can the metadata still be trusted as not to lie about its compromised state? Ironically, there is no second line of defense should the first be compromised. Our framework in this thesis shall provide trust to the integrity validation metadata, hence ensuring that our framework can be totally trusted.

b) One commonly used integrity validation metadata is the return address (Ruwase et al., 2010), (Zhou, Qin, et al., 2004) and (Zhou, Liu, et al., 2004). A common technique to guarantee the software integrity is to store the return address in a shadow memory (Serebryany et al., 2012), (Nethercote et al., 2007), (Y. J. Park et al., 2004) or shadow stack (Dang et al., 2015) .

   In essence, a shadow copy of the metadata is created, whereby a replica of the trusted return address is kept in a separate storage, for example in a hardware register. We found that, as of to date, there exists no effort to address the issue of trust for the shadow copy contents. Our proposed framework in this thesis addresses the issue of trust for shadow copy metadata via providing a mechanism for validating the integrity of the shadow copy metadata.

1.6 Summary of Contributions

The framework proposed in this thesis, termed as 'Runtime Integrity Validation of Executable Binaries' or RIBS had been developed and deployed for the integrity validation of C executables during runtime. The Trusted Platform Module (TPM) had been utilized to ensure the total trustworthiness of the executable integrity validation metadata.

In terms of effectiveness evaluations, RIBS has been extensively evaluated with three categories of attack patterns: the RIPE testsuite (Wilander et al., 2011), real world privilege escalation shellcodes and vulnerable real world applications. RIBS is successful both in the detection and mitigation of integrity violations caused by all three attack categories.

In order to gauge the performance of RIBS, the CPU time of an executable being executed as process is measured. Measurements revealed that RIBS perform better as compared to three closely related effort: CBones (Kil et al., 2007), Dytan (Clause et al., 2007) and RTC (Yong, 2004).

1.7 Thesis Organization

This thesis is organized into 5 chapters, Chapter 1: Introduction, Chapter 2: Literature Review, Chapter 3: Methodology, Chapter 4: Results and Discussion and finally Chapter 5: Conclusion and Future Work. In the paragraphs below, an executive summary of each chapter is provided.

Chapter 1 : Introduction

In this chapter, the problem statements, objectives and motivation for the presentation of our framework is presented. This chapter begins by elaborating on the problems faced by current work on validating executables, which in turn lead to the formation of our research objectives and the motivation for the development of our proposed framework.

Chapter 2 : Literature Review

In this chapter, the relevant literature on current and past efforts towards the integrity validation of executable are presented. The said efforts are grouped into five major categories related to our proposed framework. This chapter further pinpoints the shortfall of efforts under each major category.

Chapter 3 : Methodology

In this chapter the architecture and inner mechanism of our proposed framework (i.e. RIBS) is detailed. This chapter begins with an overview of the RIBS, elaborating on the design of each constituent component of RIBS and followed by the deployment of RIBS. Next, the rules used for integrity validation is also described. Subsequently, the methodologies utilized for the evaluation of RIBS in terms of effectiveness and performance are also discussed.

Chapter 4 : Results and Discussion

In this chapter, the results for both the effectiveness and performance are presented and discussed. This chapter also presents an analysis of the results obtained Further explanations were proposed as to why some rules in RIBS failed to mitigate certain attack patterns.

Chapter 5 : Conclusion and Future Work

In this chapter, we summarize our findings, outline the contributions made, point out any shortcomings and provide pointers for future expansion of our proposed framework.

The problems statements, objectives and motivation had been presented in this chapter. The next chapter, i.e. Chapter 2, shall present the literature (both past and present) which are related to our work.

REFERENCES

Alam, S., Horspool, R. N., Traore, I., & SogukPintoolsar, I. (2015). A framework for metamorphic malware analysis and real-time detection. *Computers & Security, 48*, 212-233. doi: http://dx.doi.org/10.1016/j.cose.2014.10.011

Backes, M., & Nürnberger, S. (2014). *Oxymoron: Making Fine-Grained Memory Randomization Practical by Allowing Code Sharing*. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/backes

Baratloo, A., Singh, N., and Tsai. (December, 1999). Libsafe: Protecting critical elements of stacks. http://www.research.avayalabs.com/project/libsafe/,

Bernat, A. R., & Miller, B. P. (2011). *Anywhere, any-time binary instrumentation*. Paper presented at the Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools, Szeged, Hungary.

Blacklight. (2012). Bytecode injection using ptrace. Retrieved 3rd December 2015, from https://www.youtube.com/watch?v=6ZcsoBzunUY and http://blacklight.devio.us/vidtut/index.cgi?id=

Bochs. (2015). The Bochs Emulator. Retrieved 7th Nov 2015, from, http://bochs.sourceforge.net/

Bort, J. (2015). The FBI is investigating Juniper's big, embarrassing security hole. Retrieved 18th Jan 2016, from http://www.businessinsider.my/fbi-investigates-juniper-hack-attack-2015-12/?r=US&IR=T

Brumley, D., Wang, H., Jha, S., & Song, D. (2007). *Creating Vulnerability Signatures Using Weakest Preconditions*. Paper presented at the Proceedings of the 20th IEEE Computer Security Foundations Symposium.

c0ntex. (2014). Bypassing non-executable-stack during exploitation using return-to-libc Retrieved 29th October 2015, from http://css.csail.mit.edu/6.858/2014/readings/return-to-libc.pdf

Challener, D. (2009). *A Paractical Guide to Trusted Computing*: IBM Press.

Chawla, I., & Singh, S. K. (2015). *An Automated approach for Bug Categorization using Fuzzy Logic*. Paper presented at the Proceedings of the 8th India Software Engineering Conference, Bangalore, India.

Chen, G., Jin, H., Zou, D., Zhou, B. B., Liang, Z., Zheng, W., & Shi, X. (2013). SafeStack: Automatically Patching Stack-Based Buffer Overflow Vulnerabilities. *IEEE Transactions on Dependable and Secure Computing, 10*(6), 368-379. doi: 10.1109/TDSC.2013.25

Chen, X., Slowinska, A., Andriesse, D., Bos, H., & Giuffrida, C. (2015). *StackArmor: Comprehensive Protection From Stack-based Memory Error Vulnerabilities for Binaries*. http://www.internetsociety.org/doc/stackarmor-comprehensive-protection-stack-based-memory-error-vulnerabilities-binaries

Cheng, Y., Ding, X., & Deng, R. H. (2015). *Efficient Virtualization-Based Application Protection Against Untrusted Operating System*. Paper presented at the Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, Singapore, Republic of Singapore.

Cheng, Y., Fu, X., Du, X., Luo, B., & Guizani, M. (2017). A lightweight live memory forensic approach based on hardware virtualization. *Information Sciences, 379*, 23-41. doi: http://dx.doi.org/10.1016/j.ins.2016.07.019

Chiueh, T. a. H., F. . (2001). *RAD: A Compile-Time Solution to Buffer Overflow Attacks*. Paper presented at the Proceedings of the The 21st International Conference on Distributed Computing Systems.

Christoulakis, N., Christou, G., Athanasopoulos, E., & Ioannidis, S. (2016). *HCFI: Hardware-enforced Control-Flow Integrity*. Paper presented at the Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, Louisiana, USA.

Clause, J., Li, W., & Orso, A. (2007). *Dytan: a generic dynamic taint analysis framework*. Paper presented at the Proceedings of the 2007 international symposium on Software testing and analysis, London, United Kingdom.

Consultants, P. C. (2016). The Global State of Information Security Survey 2016. Retrieved 15 September 2015., from http://www.pwc.com/gx/en/consulting-services/information-security-survey/

Cowan, C., Beattie, S., Johansen, J., & Wagle, P. (2003). *Pointguard: protecting pointers from buffer overflow vulnerabilities*. Paper presented at the Proceedings of the 12th conference on USENIX Security Symposium - Volume 12, Washington, DC.

Cowan, C., Pu, C., Maier, D., Hintony, H., Walpole, J., Bakke, P., Zhang, Q. (1998). *StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks*. Paper presented at the Proceedings of the 7th conference on USENIX Security Symposium - Volume 7, San Antonio, Texas.

Dalton, M., Kannan, H., & Kozyrakis, C. (2008). *Real-world buffer overflow protection for userspace & kernelspace*. Paper presented at the Proceedings of the 17th conference on Security symposium, San Jose, CA.

Dang, T. H. Y., Maniatis, P., & Wagner, D. (2015). *The Performance Cost of Shadow Stacks and Stack Canaries*. Paper presented at the Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, Singapore, Republic of Singapore.

Das, S., Zhang, W., & Liu, Y. (2014, 9-11 July 2014). *Reconfigurable Dynamic Trusted Platform Module for Control Flow Checking.* Paper presented at the 2014 IEEE Computer Society Annual Symposium on VLSI.

Davi, L., Liebchen, C., Sadeghi, A.-R., Snow, K. Z., & Monrose, F. (2015). *Isomeron: Code Randomization Resilient to (Just-In-Time) Return-Oriented Programming*. http://www.internetsociety.org/doc/isomeron-code-randomization-resilient-just-time-return-oriented-programming

Demsky, B. (2011). Cross-application data provenance and policy enforcement. *ACM Trans. Inf. Syst. Secur., 14*(1), 1-22. doi: 10.1145/1952982.1952988

Designer, S. (2001). Linux kernel patch from the openwall project: Non-executable user stack. from http://www.openwall.com/linux/README

Ernst, M. D., Perkins, J. H., Guo, P. J., McCamant, S., Pacheco, C., Tschantz, M. S., & Xiao, C. (2007). The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming, 69*(1–3), 35-45. doi: http://dx.doi.org/10.1016/j.scico.2007.01.015

Etoh, H. (2003). GCC Extension for Protecting Applications from Stack-Smashing Attacks. from http://www.trl.ibm.com/projects/security/ssp/

Evtyushkin, D., Elwell, J., Ozsoy, M., Ponomarev, D., Ghazaleh, N. A., & Riley, R. (2014). *Iso-X: A Flexible Architecture for Hardware-Managed Isolated Execution*. Paper presented at the Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, United Kingdom.

Fedora, P. (2015). Security Features Matrix Retrieved 27th Feb 2016, from https://fedoraproject.org/wiki/Security_Features_Matrix

Fiasco. (2016). The L4re Microkernel. Retrieved 23 Aug 2016, from http://os.inf.tu-dresden.de/fiasco/,

Garay, J. A., & Huelsbergen, L. (2006). *Software integrity protection using timed executable agents*. Paper presented at the Proceedings of the 2006 ACM Symposium on Information, computer and communications security, Taipei, Taiwan.

Gelbart, O., Leontie, E., Narahari, B., & Simha, R. (2009). A compiler-hardware approach to software protection for embedded systems. *Computers & Electrical Engineering, 35*(2), 315-328. doi: http://dx.doi.org/10.1016/j.compeleceng.2008.06.010

GNU. (2015a). The GNU clock() Retrieved 3rd December 2015, from http://www.gnu.org/software/libc/manual/html_node/CPU-Time.html

GNU. (2015b). The GNU GCC Compiler. Retrieved 19 October 2015, from https://gcc.gnu.org/

Group, T. C. (2007). *Trusted Software Stack (TSS) Specification Version 1.2 , Level 1 , Part1 Commands & Structures Manual*

Group, T. C. (2008). [Replacing Vulnerable Software with Secure Hardware].

Group, T. C. (2015). Trusted Computing Group. Retrieved 14th Nov 2015, from www.trustedcomputing.com

Gu, L., Cheng, Y., Ding, X., Deng, R. H., Guo, Y., & Shao, W. (2010). Remote Attestation on Function Execution (Work-in-Progress). In L. Chen & M. Yung (Eds.), *Trusted Systems: First International Conference, INTRUST 2009, Beijing, China, December 17-19, 2009. Revised Selected Papers* (pp. 60-72). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gu, L., Ding, X., Deng, R. H., Xie, B., & Mei, H. (2008). *Remote attestation on program execution*. Paper presented at the Proceedings of the 3rd ACM workshop on Scalable trusted computing, Alexandria, Virginia, USA.

Henderson, A., Prakash, A., Yan, L. K., Hu, X., Wang, X., Zhou, R., & Yin, H. (2014). *Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform.* Paper presented at the Proceedings of the 2014 International Symposium on Software Testing and Analysis, San Jose, CA, USA.

Hiser, J., Nguyen-Tuong, A., Co, M., Hall, M., & Davidson, J. W. (2012, 20-23 May 2012). *ILR: Where'd My Gadgets Go?* Paper presented at the 2012 IEEE Symposium on Security and Privacy.

Huffmire, T., Sherwood, T., Kastner, R., & Levin, T. (2008). Enforcing memory policy specifications in reconfigurable hardware. *Computers & Security, 27*(5–6), 197-215. doi: http://dx.doi.org/10.1016/j.cose.2008.05.002

Intel. (2015a). Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference. Retrieved 10th October 2015., from http://download.intel.com/design/intarch/manuals/24319101.pdf

Intel. (2015b). Pintoolstools – Adynamic Binary Instrumentation Tool. Retrieved 19th October 2015, from https://software.intel.com/en-us/articles/Pintools-a-dynamic-binary-instrumentation-tool

James Newsome, Dawn Song. (2005). *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software* Paper presented at the Proceedings of the 12th Annual Network and Distributed System Security Symposium.

Jin, H., Cheng, G., Zou, D., & Zhang, X. (2013). Cherub: Fine-grained application protection with on-demand virtualization. *Computers & Mathematics with Applications, 65*(9), 1326-1338. doi: http://dx.doi.org/10.1016/j.camwa.2012.02.001

Journal, L. (2015). ptrace system call. Retrieved 15th September 2015, from http://www.linuxjournal.com/article/6100

Kil3r, B. (2000). Bypassing Stackguard and Stackshield. Retrieved 11 September 2016, from http://phrack.org/issues/56/5.html

Kil, C. (2008). *Mechanisms for Protecting Software Integrity in Networked Systems.* (PhD), North Carolina State University.

Kil, C., Jun, J., Bookholt, C., Xu, J., & Ning, P. (2006, Dec. 2006). *Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software.* Paper presented at the 2006 22nd Annual Computer Security Applications Conference (ACSAC'06).

Kil, C., Sezer, E. C., Azab, A. M., Ning, P., & Zhang, X. (2009, June 29 2009-July 2 2009). *Remote attestation to dynamic system properties: Towards providing complete system integrity evidence.* Paper presented at the 2009 IEEE/IFIP International Conference on Dependable Systems & Networks.

Kil, C., Sezer, E. C., Ning, P., & Zhang, X. (2007). *Automated Security Debugging Using Program Structural Constraints.* http://dx.doi.org/10.1109/ACSAC.2007.7http://doi.ieeecomputersociety.org/10.1109/ACSAC.2007.7

Laurenzano, M., Tikir, M. M., Carrington, L., & Snavely, A. (2010). *PEBIL: Efficient static binary instrumentation for Linux.* http://dx.doi.org/10.1109/ISPASS.2010.5452024

Lee, R. B., Karig, D. K., McGregor, J. P., & Shi, Z. (2004). Enlisting Hardware Architecture to Thwart Malicious Code Injection. In D. Hutter, G. Müller, W. Stephan & M. Ullmann (Eds.), *Security in Pervasive Computing: First International Conference, Boppard, Germany, March 12-14, 2003. Revised Papers* (pp. 237-252). Berlin, Heidelberg: Springer Berlin Heidelberg.

Leontie, E. (2010). Compiler Hardware Technique for Protecting Against Buffer Overflow Attacks. https://www.seas.gwu.edu/~simha/research/HWStack.pdf

Leontie, E., Bloom, G., Narahari, B., Simha, R., & Zambreno, J. (2009). *Hardware-enforced fine-grained isolation of untrusted code*. Paper presented at the Proceedings of the first ACM workshop on Secure execution of untrusted code, Chicago, Illinois, USA.

Lin, Z., Jiang, X., Xu, D., Mao, B., & Xie, L. (2007). *AutoPaG: towards automated software patch generation with source code root cause identification and repair*. Paper presented at the Proceedings of the 2nd ACM symposium on Information, computer and communications security, Singapore.

Liu, X., Wei, Q., & Ye, Z. (2014, 8-10 Nov. 2014). *Static-Dynamic Control Flow Integrity.* Paper presented at the P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on.

Lozano, A., Mens, K., & Kellens, A. (2015). Usage contracts: Offering immediate feedback on violations of structural source-code regularities. *Science of Computer Programming, 105*, 73-91. doi: http://dx.doi.org/10.1016/j.scico.2015.01.004

Lueck, G., Patil, H., & Pereira, C. (2012). *PintoolsADX: an interface for customizable debugging with dynamic instrumentation*. Paper presented at the Proceedings of the Tenth International Symposium on Code Generation and Optimization, San Jose, California.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Hazelwood, K. (2005). Pintools: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not., 40*(6), 190-200. doi: 10.1145/1064978.1065034

Lyu, Y.-H., Hong, D.-Y., Wu, T.-Y., Wu, J.-J., Hsu, W.-C., Liu, P., & Yew, P.-C. (2014). DBILL: an efficient and retargetable dynamic binary instrumentation framework using llvm backend. *SIGPLAN Not., 49*(7), 141-152. doi: 10.1145/2674025.2576213

Mart, #237, Abadi, n., Budiu, M., #218, Erlingsson, l., & Ligatti, J. (2005). *Control-flow integrity*. Paper presented at the Proceedings of the 12th ACM conference on Computer and communications security, Alexandria, VA, USA.

McCune, J. M., Parno, B., Perrig, A., Reiter, M. K., & Seshadri, A. (2007, 20-23 May 2007). *Minimal TCB Code Execution.* Paper presented at the 2007 IEEE Symposium on Security and Privacy (SP '07).

McCune, J. M., Parno, B. J., Perrig, A., Reiter, M. K., & Isozaki, H. (2008). *Flicker: an execution infrastructure for tcb minimization*. Paper presented at the Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008, Glasgow, Scotland UK.

Mohaisen, A., Alrawi, O., & Mohaisen, M. (2015). AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security, 52*, 251-266. doi: http://dx.doi.org/10.1016/j.cose.2015.04.001

Musuvathi, M., Park, D. Y. W., Chou, A., Engler, D. R., & Dill, D. L. (2002). CMC: a pragmatic approach to model checking real code. *SIGOPS Oper. Syst. Rev., 36*(SI), 75-88. doi: 10.1145/844128.844136

Nethercote, N., & Seward, J. (2007). *Valgrind: a framework for heavyweight dynamic binary instrumentation*. Paper presented at the Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, San Diego, California, USA.

Nick L. Petroni, J., Fraser, T., Molina, J., & Arbaugh, W. A. (2004). *Copilot - a coprocessor-based kernel runtime integrity monitor*. Paper presented at the Proceedings of the 13th conference on USENIX Security Symposium - Volume 13, San Diego, CA.

Nipun Arora, H. Z., Junghwan Rhee, Kenji Yoshihira, Geoff Jiang. (2013). *iProbe: A Lightweight User-Level Dynamic Instrumentation Framework.* Paper presented at the Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13), Silicon Valley.

NIST. (2016). US CERT National Vulnerability Database. Retrieved 18th Jan 2016, from https://web.nvd.nist.gov/view/vuln/statistics-results?cves=on

Niu, B., & Tan, G. (2015). *Per-Input Control-Flow Integrity*. Paper presented at the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA.

One, A. (1998). Smashing the Stack for Fun and Profit. Retrieved 10th March 2015, from http://phrack.org/issues/49/14.html

Oragnization, M. (2016). Common vulnerabilities and exposures. . Retrieved 15 September 2015., 2015

Ozdoganoglu, H., Vijaykumar, T. N., Brodley, C. E., Kuperman, B. A., & Jalote, A. (2006). SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address. *IEEE Transactions on Computers, 55*(10), 1271-1285. doi: 10.1109/TC.2006.166

Packetstormsecurity. (2016). Ignore Security and It'll Go Away. Retrieved 6th Sept 2015, from https://packetstormsecurity.com/

Pappas, V., Polychronakis, M., & Keromytis, A. D. (2012, 20-23 May 2012). *Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization*. Paper presented at the 2012 IEEE Symposium on Security and Privacy.

Park, T., & Shin, K. G. (2005). Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing, 4*(3), 297-309. doi: 10.1109/tmc.2005.44

Park, Y. J., & Lee, G. (2004). *Repairing return address stack for buffer overflow protection*. Paper presented at the Proceedings of the 1st conference on Computing frontiers, Ischia, Italy.

Paulson, L. D. (2004). New chips stop buffer overflow attacks. *37* (10), 28-30.

PAX. (2001). PaX.

Pham, V.-T., B, M., #246, hme, & Roychoudhury, A. (2016). *Model-based whitebox fuzzing for program binaries*. Paper presented at the Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, Singapore.

Piromsopa, K., & Enbody, R. J. (2006). Secure Bit: Transparent, Hardware Buffer-Overflow Protection. *IEEE Transactions on Dependable and Secure Computing, 3*(4), 365-376. doi: 10.1109/TDSC.2006.56

Plank, J. (2008). CS360 Lecture Notes Retrieved 27th Feb 2016, from http://web.eecs.utk.edu/~mbeck/classes/cs360/360/notes/Setjmp/lecture.html

Prasad, M., & Chiueh, T.-c. (2003). *A Binary Rewriting Defense Against Stack based Buffer Overflow Attacks*. http://www.usenix.org/events/usenix03/tech/prasad.html

Qin, F., Wang, C., Li, Z., Kim, H.-s., Zhou, Y., & Wu, Y. (2006). *LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks*. Paper presented at the Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture.

Roy, A., Hand, S., & Harris, T. (2011). Hybrid binary rewriting for memory access instrumentation. *SIGPLAN Not., 46*(7), 227-238. doi: 10.1145/2007477.1952711

Ruwase, O., Chen, S., Gibbons, P. B., & Mowry, T. C. (2010). *Decoupled lifeguards: enabling path optimizations for dynamic correctness checking tools*. Paper presented at the Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, Toronto, Ontario, Canada.

Sahay, S. K., & Sharma, A. (2016). GrouPintoolsg the Executables to Detect Malwares with High Accuracy. *Procedia Computer Science, 78*, 667-674. doi: http://dx.doi.org/10.1016/j.procs.2016.02.115

Sailer, R., Zhang, X., Jaeger, T., & Doorn, L. v. (2004). *Design and implementation of a TCG-based integrity measurement architecture*. Paper presented at the Proceedings of the 13th conference on USENIX Security Symposium - Volume 13, San Diego, CA.

Sailer, R., Zhang, X., Jaeger, T., & Doorn, L. v. (2015). The IMA Wiki Retrieved 14th Nov 2015, from http://sourceforge.net/p/linux-ima/wiki/Home/

Selorst, M. (2011). TrustedGRUB Secure Bootloader. Retrieved 20 November 2015, from, http://sourceforge.net/projects/trustedgrub

Serebryany, K., Bruening, D., Potapenko, A., & Vyukov, D. (2012). *AddressSanitizer: a fast address sanity checker*. Paper presented at the Proceedings of the 2012 USENIX conference on Annual Technical Conference, Boston, MA.

Seshadri, A., Luk, M., Shi, E., Perrig, A., Doorn, L. v., & Khosla, P. (2005). *Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems*. Paper presented at the Proceedings of the twentieth ACM symposium on Operating systems principles, Brighton, United Kingdom.

Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., & Boneh, D. (2004). *On the effectiveness of address-space randomization*. Paper presented at the Proceedings of the 11th ACM conference on Computer and communications security, Washington DC, USA.

Shaneck, M., Mahadevan, K., Kher, V., & Kim, Y. (2005). Remote Software-Based Attestation for Wireless Sensors. In R. Molva, G. Tsudik & D. Westhoff (Eds.), *Security and Privacy in Ad-hoc and Sensor Networks: Second European Workshop, ESAS 2005, Visegrad, Hungary, July 13-14, 2005. Revised Selected Papers* (pp. 27-41). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sherei, S. E. (2016). Return to LibC Tutorial. Retrieved 27th Feb 2016, from http://www.elsherei.com/?page_id=229

Singh, S., Estan, C., Varghese, G., & Savage, S. (2004). *Automated worm fingerprinting*. Paper presented at the Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6, San Francisco, CA.

Sridhar, S., Shapiro, J. S., Northup, E., & Bungale, P. P. (2006). *HDTrans: an open source, low-level dynamic instrumentation system*. Paper presented at the Proceedings of the 2nd international conference on Virtual execution environments, Ottawa, Ontario, Canada.

Stallman, R. (Writer). (2015). GCC Compiler Manual version 4.8.5: Free Software Foundation, Inc.

Stergiopoulos, G., Katsaros, P., & Gritzalis, D. (2017). Program analysis with risk-based classification of dynamic invariants for logical error detection. *Computers & Security*. doi: http://dx.doi.org/10.1016/j.cose.2017.02.007

Tang, A., Sethumadhavan, S., & Stolfo, S. (2015). *Heisenbyte: Thwarting Memory Disclosure Attacks using Destructive Code Reads*. Paper presented at the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA.

Tenouk. (2015). Stack Frame Construction During Function Calls. Retrieved 10th October 2015, from http://www.tenouk.com/Bufferoverflowc/Bufferoverflow2.html

Tian, D., Xiong, X., Hu, C., & Liu, P. (2014). Defeating buffer overflow attacks via virtualization. *Computers & Electrical Engineering, 40*(6), 1940-1950. doi: http://dx.doi.org/10.1016/j.compeleceng.2013.11.032

Trousers. (2009). Trousers TPM Library. Retrieved 3rd December 2015, from http://trousers.sourceforge.net/

Tuck, N., Calder, B., & Varghese, G. (2004, 04-08 Dec. 2004). *Hardware and Binary Modification Support for Code Pointer Protection From Buffer Overflow*. Paper presented at the Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on.

University, S. (2015). The TPM Architecture and Features. Retrieved 10 November 2015, from https://crypto.stanford.edu/cs155old/cs155-spring06/08-TCG.pdf

Vendicator. (2001). StackShield: A 'Stack Smashing' Technique Protection Tool for Linux,. Retrieved 29th October 2015, from http://www.angelfire.com/sk/stackshield/download.html

Vishwath Mohan and , P. L. (2015). , *Opaque Control Flow Integrity*. Paper presented at the NDSS'15.

Wang, H. J., Guo, C., Simon, D. R., & Zugenmaier, A. (2004). Shield: vulnerability-driven network filters for preventing known vulnerability exploits. *SIGCOMM Comput. Commun. Rev., 34*(4), 193-204. doi: 10.1145/1030194.1015489

Wang, P., & Wang, Y.-S. (2015). Malware behavioural detection and vaccine development by using a support vector model classifier. *Journal of Computer and System Sciences, 81*(6), 1012-1026. doi: http://dx.doi.org/10.1016/j.jcss.2014.12.014

Wang, T., Song, C., & Lee, W. (2014). Diagnosis and Emergency Patch Generation for Integer Overflow Exploits. In S. Dietrich (Ed.), *Detection of Intrusions and Malware, and Vulnerability Assessment: 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings* (pp. 255-275). Cham: Springer International Publishing.

Wang, X., Pan, C.-C., Liu, P., & Zhu, S. (2006). *SigFree: a signature-free buffer overflow attack blocker*. Paper presented at the Proceedings of the 15th conference on USENIX Security Symposium - Volume 15, Vancouver, B.C., Canada.

Wartell, R., Mohan, V., Hamlen, K. W., & Lin, Z. (2012). *Securing untrusted code via compiler-agnostic binary rewriting*. Paper presented at the Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, Florida, USA.

Wei, M., Wagner, S., Hellman, R., & Wessel, S. (2014, 24-26 Sept. 2014). *Integrity Verification and Secure Loading of Remote Binaries for Microkernel-Based Runtime Environments*. Paper presented at the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications.

Wilander, J., Nikiforakis, N., Younan, Y., Kamkar, M., & Joosen, W. (2011). *RIPE: runtime intrusion prevention evaluator*. Paper presented at the Proceedings of the 27th Annual Computer Security Applications Conference, Orlando, Florida, USA.

Wojtczuk, R. (1998). Defeating Solar Designer's Non-executable Stack Patch. Retrieved 11 September 2016, from http://insecure.org/sploits/non-executable.stack.problems.html

Xu, J., Kalbarczyk, Z., Patel, S., and Iyer, R.K. (2002). *Architecture Support for Defending against Buffer Overflow Attacks.* Paper presented at the Procedeings on Workshop Evaluating and Architecting System Dependability

Yong, S. H. (2004). *Runtime Monitoring of C Programs for Security and Correctness,.* (PhD), University of Wisconsin Madison.

Yong, S. H., & Horwitz, S. (2003). Protecting C programs from attacks via invalid pointer dereferences. *SIGSOFT Softw. Eng. Notes, 28*(5), 307-316. doi: 10.1145/949952.940113

Zeng, B., Tan, G., & Morrisett, G. (2011). *Combining control-flow integrity and static analysis for efficient and validated data sandboxing*. Paper presented at the Proceedings of the 18th ACM conference on Computer and communications security, Chicago, Illinois, USA.

Zeng, J., Fu, Y., & Lin, Z. (2015). *PEMU: A Pintools Highly Compatible Out-of-VM Dynamic Binary Instrumentation Framework*. Paper presented at the Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Istanbul, Turkey.

Zhang, C., Wei, T., Chen, Z., Duan, L., Szekeres, L., McCamant, S., Zou, W. (2013, 19-22 May 2013). *Practical Control Flow Integrity and Randomization for Binary Executables.* Paper presented at the Security and Privacy (SP), 2013 IEEE Symposium on.

Zhang, M., Qiao, R., Hasabnis, N., & Sekar, R. (2014). A platform for secure static binary instrumentation. *SIGPLAN Not., 49*(7), 129-140. doi: 10.1145/2674025.2576208

Zhang, M., & Sekar, R. (2013). *Control Flow Integrity for COTS Binaries*. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/Zhang

Zhou, P., Liu, W., Fei, L., Lu, S., Qin, F., Zhou, Y.,Torrellas, J. (2004). *AccMon: Automatically Detecting Memory-Related Bugs via Program Counter-Based Invariants*. Paper presented at the Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, Portland, Oregon.

Zhou, P., Qin, F., Liu, W., Zhou, Y., & Torrellas, J. (2004). *iWatcher: Efficient Architectural Support for Software Debugging*. Paper presented at the Proceedings of the 31st annual international symposium on Computer architecture, Munchen, Germany.

Zhu, E., Liu, F., Wang, Z., Liang, A., Zhang, Y., Li, X., & Li, X. (2015). Dytaint: The implementation of a novel lightweight 3-state dynamic taint analysis framework for x86 binary programs. *Computers & Security, 52*, 51-69. doi: http://dx.doi.org/10.1016/j.cose.2015.03.008