

**HARDWARE-ASSISTED SECURE EXECUTION
ON REMOTE, UNTRUSTED SYSTEMS**

SANDEEP KUMAR



AMAR NATH AND SHASHI KHOSLA SCHOOL OF INFORMATION
TECHNOLOGY

INDIAN INSTITUTE OF TECHNOLOGY DELHI

JULY 2024

©Indian Institute of Technology Delhi (IITD), New Delhi, 2024

**HARDWARE-ASSISTED SECURE EXECUTION
OF PROGRAMS ON UNTRUSTED
PLATFORMS**

by

SANDEEP KUMAR

AMAR NATH AND SHASHI KHOSLA SCHOOL OF INFORMATION
TECHNOLOGY

Submitted

in fulfillment of the requirements of the degree of **Doctor of Philosophy**

to the



Indian Institute of Technology Delhi

JULY 2024

For my grandfather,
Late Sh. Mahendra Prasad

Certificate

This is to certify that the thesis titled **HARDWARE-ASSISTED SECURE EXECUTION OF PROGRAMS ON UNTRUSTED PLATFORMS** being submitted by **Mr. Sandeep Kumar** for the award of **Doctor of Philosophy in Information Technology** is a record of bona fide work carried out by him under my guidance and supervision at the **Amar Nath and Shashi Khosla School of Information Technology, Indian Institute of Technology Delhi**. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Smruti Ranjan Sarangi

Professor

Department of Computer Science and Engineering

Indian Institute of Technology Delhi

New Delhi- 110016

Acknowledgements

I had a great experience during my Ph.D. and will cherish it for the rest of my life. I learned great skills that made me a better person and researcher, and I made a few lifelong friends.

I want to start by thanking my Ph.D. supervisor, **Dr. Smruti R. Sarangi**. I am grateful for his constant guidance, which has made me a better researcher. I also deeply appreciate his push to understand the fundamentals of any problem before starting a project. This one piece of advice has helped me with all of my projects. I also loved our conversation about life and different non-Ph.D.-related topics. He taught me the importance of hard work. I will not forget working with him at two in the morning to meet a paper submission deadline. I thank him again. I also thank my student research committee members, **Prof. Anshul Kumar**, **Prof. Kolin Paul**, and **Prof. Shouribrata Chatterjee**, for providing valuable feedback on my research.

I also thank my master's advisor, **Prof. K. Gopinath**. He was gracious enough to accept me as his student and guided me during my master's and my stint as a research assistant. I will be forever grateful to him for his guidance and support.

I am grateful to the administrative staff in the department: Mr. Rajesh Kumar, Mr. Suresh, Ms. Vandana, Ms. Manju, and Ms. Aditi. I am especially grateful to Ms. Aditi. She went beyond her duties to help me fulfill the formalities for various things.

During the course of my Ph.D., I found many great friends: Dr. Arindam Bhattacharya, Dilpreet Kaur, Dr. Ovia Shesadri, Dr. Dishant Goel, Dr. Vinayak Gupta, Dr. Diksha Moolchandani, Dr. Priyanka Singla, Dr. Anupam Sobti, Omais Shafi, Saurabh Tewari, Dr. Shubhankar Suman, Dr. Hameedah Sultan, Dr. Janibul Bashit, Abhisek Panda, and Rahul Kanyal.

I am deeply grateful to Dr. Diksha Moolchandani for collaborating on multiple projects and fruitful lab discussions. I learned many things from her regarding resilience, technical background, and general research direction. I would also like to thank Abhisek Panda. We collaborated on many projects, and I will be grateful for his help in project discussions, experiment setup, and finalizing the projects. He is a great person, friend, and researcher. I wish him all the

best for his Ph.D. journey. I thank Arindam, Dilpreet, Ovia, Dishant, Vinayak, Sikha, and Anupam for those fun lab discussions on research and life, impromptu lunches, and walks around the campus.

I would also like to thank Kaushal Bisht and Oindrila Mitra for being there with me through the Ph.D. journey and for all those house parties that acted as a stress buster and motivated me. I am deeply thankful to them. Apart from them, I would like to thank Akshar Kaul, Remish Minz, Chandrahas Dewangan, Rakesh Malviya, Rahul Sharma, Ankit Dixit, Saurabh Mondal, Kiran Prasad, Anirban Laha, and Suvam Mukherjee for being a part of my life at different phases and making it fun to live. I thank Aditi Puri for the constant push to submit the thesis.

I was fortunate to complete a 7-month internship at Intel Labs in Bangalore, India. I want to thank Aravinda Prasad and Sreenivas Subramoney for giving me the opportunity and their constant guidance during the internship, which helped me grow as a researcher.

Finally, I would like to thank my parents, Mr. Ramjee Thakur and Mrs. Rema Thakur, for their unconditional love and support throughout my life. Nothing would have been possible without them. I also thank my brother, Dr. Nilesh Kumar, for his constant life, fitness lessons, and all those fun moments we shared. I am deeply grateful for the love and support I received from my grandparents, uncles, aunts, and cousins. I especially thank my grandfather, Mr. Mahendra Prasad, for his immense love, support, and belief that I can achieve anything. I wish he were here to see this.

Sandeep Kumar

Abstract

Securing code and data in a cloud computing environment is a quintessential problem. Cloud-based remote systems are untrusted, and thus, there is a need to ensure their trustworthiness. Moreover, many such applications may be dealing with sensitive data, which increases the incentive for a malicious adversary to try to subvert the execution flow.

The traditional, software-only solution to protect access to code and data is to encrypt them when they are resident in secondary storage media such as the hard disk or to use special access-controlling functions such as license managers that check the existence of a key with the client machine. In this thesis, we show that such software-only solutions, using a provable, strong, unbreakable cryptographic primitive, are vulnerable to a malicious entity with admin-level privileges. The ingenuity of such an attack is that it does not interfere with the functioning of the license check manager method but rather with its outcome, rendering the entire defense method ineffective. We designed a novel *control flow bending* attack on the execution of popular open-source software using their control flow graph (or CFG) representation and broke the license check manager of many commercial applications, including MySQL.

The community in this space has also proposed hardware-based defense solutions where the execution platform guarantees the security of an application. Although effective, pure hardware-based solutions have a long development cycle and are not easily patchable in case of a new attack. Hence, *hybrid methods*, where hardware and software work in tandem, are gaining popularity. In this setting, the hardware does the heavy lifting and exposes a common interface to gain insight into the execution of an application. The information obtained from these modules is then processed in the software to provide security to an application.

In this thesis, our main focus is to ensure secure execution on a remote, untrusted server. Hence, we focus on trusted execution environments (or TEEs) designed to ensure an application's security in the presence of a privileged attacker. In a TEE, the hardware ensures the code and data's confidentiality, integrity, and freshness. We selected Intel Software Guard eXtension (or Intel SGX), a TEE solution from Intel, for our experiments due to its wide availability and

popularity. In SGX, the code and data remain encrypted on the disk and the main memory and are only decrypted when brought into the chip. This approach ensures protection from privileged attackers, such as a malicious operating system (OS) or hypervisor. However, there are certain challenges here. As the OS is not a part of the trusted computing base (TCB), the “secure” part of applications cannot make system calls, issue certain instructions, or take advantage of shared libraries.

We start by addressing a basic requirement for working with Intel SGX: developing a *standard benchmark suite* for evaluating the performance of applications built on SGX. Prior work in this area used workloads that did not provide a holistic view of the execution characteristics of the SGX. We created a suite of applications from different domains showing different characteristics in terms of CPU usage and memory usage to gain an in-depth insight into the workings of SGX. In order to measure the efficiency of our proposed benchmark suite, we also define a scoring method to gauge the quality of a benchmark suite. The definition of these scores is based on a list of desirable features that have been qualitatively enunciated in prior work (notably by the authors of the Parsec benchmarks). We use the scores to show that our benchmark suite for SGX (SGXGauge) qualitatively outperforms other suites in this domain. Furthermore, using our novel set of metrics, we were able to critically analyze the pros and cons of the following benchmark suites and explain qualitative observations made by other authors more quantitatively: SPEC, Parsec, and Ligra.

Subsequently, we propose a novel approach to prevent control-flow bending (or CFB) attacks by leveraging SGX. Our approach is to “*handicap*” a binary by placing a small portion of secure functions – the license check and its forward pass – inside Intel SGX. Access to the secure function is only allowed after successfully validating the license file within SGX. Attempting to bypass the license check manager (as in a CFB attack) will result in a crash. Although secure, this approach has a high overhead of checking license checks due to a remote attestation step for every access to a secure function. To address this performance issue, we designed a novel solution to reduce the cost of multiple license checks by enabling a partial number of authentication requests to be processed locally. Next, we enabled access to an SGX-optimized file system for an application running within SGX. Our design uses a combination of encryption and hashing to ensure the confidentiality, integrity, and freshness of the data blocks written to the file system. Our design maintains a small cache of data and metadata blocks to ensure fast read and write operations.

As our final contribution, we combine all the previous methods and propose a novel method to enable live migration of large SGX enclaves from one machine to another. The first version of SGX supported a secure, usable memory of 92 MB. However, the second generation of SGX

called *scalable SGX*, can support secure memory up to 512 GB. Prior work solved the migration challenge by using the traditional stop-and-copy approach. However, this approach fails to scale to the gigabytes of secure memory in the scalable SGX. We identified key challenges in adopting a live migration scheme for secure enclaves. We propose a novel solution that addresses these challenges and enables secure live migration of a large SGX enclave from one machine to another.

संक्षेप

क्लाउड कंप्यूटिंग वातावरण में कोड और डेटा सुरक्षित करना एक सर्वोत्कृष्ट समस्या है। बादल-आधारित रिमोट सिस्टम अविश्वसनीय हैं, और इस प्रकार उनकी विश्वसनीयता सुनिश्चित करने की आवश्यकता है। इसके अलावा, ऐसे कई एप्लिकेशन संवेदनशील डेटा से निपट सकते हैं - इससे निष्पादन प्रवाह को नष्ट करने की कोशिश में दुर्भावनापूर्ण प्रतिद्वंद्वी का प्रोत्साहन बढ़ जाता है।

कोड और डेटा तक पहुंच की सुरक्षा के लिए पारंपरिक, सॉफ्टवेयर-केवल समाधान उन्हें एन्क्रिप्ट करना है जब वे हार्ड डिस्क जैसे द्वितीयक भंडारण मीडिया में रहते हैं, या लाइसेंस प्रबंधकों जैसे विशेष एक्सेस-नियंत्रण कार्यों का उपयोग करते हैं जो डेटा के अस्तित्व की जांच करते हैं। क्लाइंट मशीन के साथ कुंजी। इस थीसिस में, हम दिखाते हैं कि ऐसे सॉफ्टवेयर-केवल समाधान, एक सिद्ध मजबूत, अटूट क्रिप्टोग्राफिक प्रिमिटिव का उपयोग करते हुए, व्यवस्थापक-स्तर के विशेषाधिकारों के साथ एक दुर्भावनापूर्ण इकाई के प्रति संवेदनशील होते हैं। इस तरह के हमले की सरलता यह है कि यह लाइसेंस जांच प्रबंधक पद्धति के कामकाज में हस्तक्षेप नहीं करता है, बल्कि इसके परिणाम के साथ, संपूर्ण रक्षा पद्धति को अप्रभावी बना देता है। हमने उनके नियंत्रण प्रवाह ग्राफ (या सीएफजी) प्रतिनिधित्व का उपयोग करके लोकप्रिय ओपन सोर्स सॉफ्टवेयर के निष्पादन पर एक नया नियंत्रण प्रवाह झुकने वाला हमला डिज़ाइन किया और MySQL सहित कई वाणिज्यिक अनुप्रयोगों के लाइसेंस जांच प्रबंधक को तोड़ दिया।

इस क्षेत्र में समुदाय ने हार्डवेयर-आधारित रक्षा समाधान भी प्रस्तावित किया है जहां निष्पादन प्लेटफॉर्म किसी एप्लिकेशन की सुरक्षा की गारंटी देता है। हालांकि प्रभावी, शुद्ध हार्डवेयर-आधारित समाधानों का विकास चक्र लंबा होता है और नए हमले की स्थिति में इन्हें आसानी से पैच नहीं किया जा सकता है। इसलिए, हाइब्रिड तरीके, जहां हार्डवेयर और सॉफ्टवेयर मिलकर काम करते हैं, लोकप्रियता हासिल कर रहे हैं। इस सेटिंग में, हार्डवेयर भारी भारोत्तोलन करता है और किसी एप्लिकेशन के निष्पादन में अंतर्दृष्टि प्राप्त करने के लिए एक सामान्य इंटरफ़ेस को उजागर करता है। इन मॉड्यूल से प्राप्त जानकारी को किसी एप्लिकेशन को सुरक्षा प्रदान करने के लिए सॉफ्टवेयर में संसाधित किया जाता है।

इस थीसिस में, हमारा मुख्य फोकस दूरस्थ, अविश्वसनीय सर्वर पर सुरक्षित निष्पादन सुनिश्चित करना है। इसलिए, हम विश्वसनीय निष्पादन वातावरण (या टीईई) पर ध्यान केंद्रित करते हैं जो एक विशेषाधिकार प्राप्त हमलावर की उपस्थिति में एप्लिकेशन की सुरक्षा सुनिश्चित करने के लिए डिज़ाइन किया गया है। टीईई में, हार्डवेयर कोड और डेटा की गोपनीयता, अखंडता और ताजगी सुनिश्चित करता है। हमने इसकी व्यापक उपलब्धता और लोकप्रियता के कारण अपने प्रयोगों के लिए इंटेल से एक टीईई समाधान इंटेल सॉफ्टवेयर गार्ड एक्सटेंशन (या इंटेल एसजीएक्स) का चयन किया। एसजीएक्स में, कोड और डेटा डिस्क और मुख्य मेमोरी पर एन्क्रिप्टेड रहते हैं और चिप में लाए जाने पर ही डिक्रिप्ट होते हैं। यह दृष्टिकोण दुर्भावनापूर्ण ऑपरेटिंग सिस्टम (ओएस) या हाइपरवाइजर जैसे

विशेषाधिकार प्राप्त हमलावरों से सुरक्षा सुनिश्चित करता है। हालाँकि, यहाँ कुछ चुनौतियाँ हैं। चूंकि ओएस विश्वसनीय कंप्यूटिंग बेस (टीसीबी) का हिस्सा नहीं है, इसलिए एप्लिकेशन का "सुरक्षित" हिस्सा सिस्टम कॉल नहीं कर सकता, कुछ निर्देश जारी नहीं कर सकता और साझा लाइब्रेरी का लाभ नहीं उठा सकता।

हम इंटेल एसजीएक्स के साथ काम करने के लिए एक बुनियादी आवश्यकता को संबोधित करते हुए शुरुआत करते हैं: एसजीएक्स पर निर्मित अनुप्रयोगों के प्रदर्शन के मूल्यांकन के लिए एक मानक बेंचमार्क सूट विकसित करना। इस क्षेत्र में पहले के काम में ऐसे कार्यभार का उपयोग किया गया था जो एसजीएक्स की निष्पादन विशेषताओं का समग्र दृष्टिकोण प्रदान नहीं करता था। हमने एसजीएक्स के कामकाज में गहराई से जानकारी हासिल करने के लिए सीपीयू उपयोग और मेमोरी उपयोग के संदर्भ में विभिन्न विशेषताओं को दिखाने वाले विभिन्न डोमेन से अनुप्रयोगों का एक सूट बनाया। हमारे प्रस्तावित बेंचमार्क सूट की दक्षता को मापने के लिए, हम बेंचमार्क सूट की गुणवत्ता को मापने के लिए एक स्कोरिंग विधि भी परिभाषित करते हैं। इन अंकों की परिभाषा वांछनीय विशेषताओं की एक सूची पर आधारित है जिन्हें पिछले काम में गुणात्मक रूप से प्रतिपादित किया गया है (विशेषकर पारसेक बेंचमार्क के लेखकों द्वारा)। हम यह दिखाने के लिए स्कोर का उपयोग करते हैं कि SGX (SGXGauge) के लिए हमारा बेंचमार्क सूट, गुणात्मक रूप से इस डोमेन में अन्य सुइट्स से बेहतर प्रदर्शन करता है। इसके अलावा, मेट्रिक्स के हमारे नए सेट का उपयोग करके, हम निम्नलिखित बेंचमार्क सुइट्स के पेशेवरों और विपक्षों का गंभीर रूप से विश्लेषण करने और अन्य लेखकों द्वारा की गई गुणात्मक टिप्पणियों की व्याख्या करने में सक्षम थे।

अधिक मात्रात्मक रूप से: स्पेक, पारसेक और लिग्रा ।

इसके बाद, हम एसजीएक्स का लाभ उठाकर नियंत्रण-प्रवाह झुकने (या सीएफबी) हमलों को रोकने के लिए एक नया दृष्टिकोण प्रस्तावित करते हैं। हमारा दृष्टिकोण इंटेल एसजीएक्स के अंदर सुरक्षित कार्यों - लाइसेंस जांच और इसके फॉरवर्ड पास - का एक छोटा सा हिस्सा रखकर बाइनरी को "विकलांग" करना है। एसजीएक्स के भीतर लाइसेंस फ़ाइल को सफलतापूर्वक मान्य करने के बाद ही सुरक्षित फ़ंक्शन तक पहुंच की अनुमति दी जाती है। लाइसेंस जांच प्रबंधक को बायपास करने का प्रयास (सीएफबी हमले के रूप में) के परिणामस्वरूप दुर्घटना होगी। हालांकि सुरक्षित, इस दृष्टिकोण में एक सुरक्षित फ़ंक्शन तक प्रत्येक पहुंच के लिए दूरस्थ सत्यापन चरण के कारण लाइसेंस जांच की जांच करने का एक उच्च ओवरहेड है। इस प्रदर्शन समस्या का समाधान करने के लिए, हमने आंशिक संख्या में प्रमाणीकरण अनुरोधों को स्थानीय स्तर पर संसाधित करने को सक्षम करके एकाधिक लाइसेंस जांच की लागत को कम करने के लिए एक नया समाधान तैयार किया है। इसके बाद, हमने SGX के भीतर चल रहे एप्लिकेशन के लिए SGX-अनुकूलित फ़ाइल सिस्टम तक पहुंच सक्षम की। हमारा डिज़ाइन फ़ाइल सिस्टम में लिखे गए डेटा ब्लॉक की गोपनीयता, अखंडता और ताजगी सुनिश्चित करने के लिए एन्क्रिप्शन और हैशिंग के संयोजन का उपयोग करता है। हमारा डिज़ाइन तेजी से पढ़ने और लिखने के संचालन को सुनिश्चित करने के लिए डेटा और मेटाडेटा ब्लॉक का एक छोटा कैश बनाए रखता है।

अपने अंतिम योगदान के रूप में, हम सभी पिछले तरीकों को जोड़ते हैं और बड़े एसजीएक्स एन्क्लेव के एक मशीन से दूसरे मशीन में लाइव माइग्रेशन को सक्षम करने के लिए एक उपन्यास विधि का प्रस्ताव करते हैं। एसजीएक्स का पहला संस्करण 92 एमबी की सुरक्षित, उपयोग योग्य मेमोरी का समर्थन करता था। हालाँकि, SGX की दूसरी पीढ़ी, जिसे स्केलेबल SGX कहा जाता है, 512 जीबी तक सुरक्षित मेमोरी का समर्थन कर सकती है। पिछले कार्य ने पारंपरिक स्टॉप-एंड-कॉपी दृष्टिकोण का उपयोग करके प्रवासन चुनौती को हल किया। हालाँकि, यह दृष्टिकोण स्केलेबल SGX में मौजूद गीगाबाइट सुरक्षित मेमोरी को स्केल करने में विफल रहता है। हमने सुरक्षित एन्क्लेव के लिए लाइव माइग्रेशन योजना को अपनाने में आने वाली प्रमुख चुनौतियों की पहचान की और एक उपन्यास समाधान का प्रस्ताव दिया जो इन चुनौतियों का समाधान करता है और एक बड़े एसजीएक्स एन्क्लेव के एक मशीन से दूसरे मशीन में सुरक्षित लाइव माइग्रेशन को सक्षम बनाता है।

Contents

Certificate	i
Acknowledgements	iii
Abstract	v
List of Figures	xxi
List of Tables	xxix
1 Introduction	1
1.1 Stage I: Setup	4
1.2 Stage II: Securing a Single Application	6
1.3 Stage III: End-to-End Security	8
2 Survey of Hardware-Assisted Mechanisms to Enforce Control Flow Integrity	11
2.1 Introduction	11
2.1.1 Scope of the Survey	13
2.1.2 Organization of the Chapter	14
2.2 Background	14

2.2.1	Safe and unsafe languages	15
2.2.2	Attacks on control flow integrity	16
2.2.3	Hardware advancements	20
2.2.4	Hardware-accelerated instructions	24
2.2.5	Hardware performance counters	24
2.2.6	RISC-V	25
2.3	Related work	27
2.4	Taxonomy	29
2.4.1	Challenges	29
2.4.2	Classification	30
2.4.3	<i>A binary-based</i> Classification	30
2.4.4	<i>A process-based</i> Classification	31
2.4.5	<i>A system-based</i> Classification	31
2.5	Binary-based defense methods	32
2.5.1	Binary re-writing	33
2.5.2	Binary splitting: secure and unsecure	35
2.5.3	Binary randomization	36
2.5.4	Control flow tracking of binaries	39
2.5.5	Heterogeneous architectures	41
2.6	Process-based defense methods	42
2.6.1	Dynamic process randomization	45
2.6.2	Protecting the stack	48
2.6.3	Runtime metadata	57

2.6.4	Protecting control flow data	60
2.7	System-based defense methods	63
2.7.1	Control flow verification	65
2.7.2	Application execution trace	68
2.7.3	Trusted services	71
2.7.4	Machine/Deep learning approaches	72
2.7.5	Hardware performance counters	75
2.8	Research Directions and Future Work	77
2.8.1	Bounding the Problem	78
2.8.2	Configuring Defense Methods	78
2.8.3	What about the Operating System?	79
2.8.4	Future of Computing and Security Needs	80
2.9	Conclusion	80
3	SGXGauge: A Comprehensive Benchmark Suite for Intel SGX	83
3.1	Introduction	83
3.2	Background	85
3.2.1	Intel SGX	85
3.2.2	Enclave Page Cache	86
3.2.3	Enclave Transitions: ECALLs and OCALLs	86
3.2.4	Executing complete application in SGX	87
3.2.5	Library Operating Systems	88
3.3	Related Work and Motivation	89
3.3.1	Related Work	89

3.3.2	Motivation	90
3.4	SGXGauge Benchmark Suite	93
3.4.1	Evaluation Modes and Input Settings	94
3.4.2	Workloads' Description	96
3.4.3	Porting to Intel SGX	98
3.4.4	Running on GrapheneSGX	98
3.5	Evaluation	99
3.5.1	Experimental Setup	99
3.5.2	Evaluation Plan	100
3.5.3	<i>Native</i> mode Performance	102
3.5.4	<i>LibOS</i> Mode Performance	103
3.5.5	<i>Native</i> Mode vs <i>LibOS</i> Mode	104
3.5.6	Switchless Mode	104
3.5.7	Intel SGX Latencies	104
3.5.8	<i>Native</i> Mode Performance	105
3.5.9	Counter Impact on Performance	107
3.5.10	GrapheneSGX start-up overhead	108
3.5.11	What About I/O?	109
3.6	Conclusion	111
4	Perspector: Benchmarking Benchmark Suites	113
4.1	Introduction	113
4.2	Related Work	115
4.3	Benchmark Quality Metrics	116

4.3.1	Diversification: Cluster Score	117
4.3.2	Phase changes: Trend Score	118
4.3.3	Coverage Score	120
4.3.4	Spread Score	122
4.4	Evaluation	123
4.4.1	Benchmark Suites' Scores	125
4.4.2	Focused Scoring	126
4.4.3	Benchmark Suite Subset Generation	127
4.5	Conclusion	129
5	F-LaaS: A Control-Flow-Attack Immune License-as-a-Service Model	131
5.1	Introduction	131
5.2	Background: License Managers and Attacks	133
5.2.1	Design of a License Manager	133
5.2.2	Control Flow Bending Attacks	134
5.2.3	CFDA and CGA	134
5.2.4	Control Flow Graphs (CFGs)	136
5.3	Threat and Security Model	136
5.3.1	Threat Model	136
5.3.2	Security Model	137
5.4	Characteristics of a License Manager	137
5.4.1	Creation of the CFG	137
5.4.2	Clustering of the CFG	138
5.4.3	Classification of Nodes and Clusters	139

5.4.4	Key Assumptions	139
5.5	Algorithm to Prune the CFG	140
5.5.1	Radius-based pruning	141
5.5.2	Score-based pruning	141
5.5.3	Degree-based pruning	142
5.6	Defense Method: <i>F-LaaS</i> : Flexible- Licensing as a Service	143
5.6.1	Selection of the <i>O</i> -Functions	144
5.7	Evaluation	145
5.7.1	Benchmark Details	145
5.7.2	Experimental Setup	145
5.7.3	Hyper-parameter Tuning	145
5.7.4	Validation of Assumptions	147
5.7.5	Results for <i>SmartCFB</i>	148
5.7.6	Selection of the <i>O</i> -function(s)	149
5.8	Conclusion	150
6	SecureLease: Maintaining Execution Control in The Wild using Intel SGX	151
6.1	Introduction	151
6.2	Background and Motivation	154
6.2.1	Software-based Authentication Modules	154
6.2.2	FaaS- and Plugin-based Applications	156
6.2.3	Intel SGX	157
6.3	Related Work	159
6.3.1	Takeaways	160

6.4	Key Design Principles	161
6.4.1	Threat Model	161
6.4.2	Dependency-based Partitioning	161
6.4.3	Modeling Lease Types	163
6.4.4	Lease Management	164
6.5	Implementation	165
6.5.1	SL-Remote & SL-Manager	165
6.5.2	SL-Local	165
6.5.3	Adaptive GCL Renewal	169
6.5.4	Issuing a Lease	170
6.5.5	Committing a Lease	171
6.5.6	SL-Local Exit and Re-Initialization	172
6.5.7	Replay Attacks on SL-Local	172
6.5.8	Design Benefits	173
6.6	Security Analysis	173
6.6.1	Control Flow Bending (CFB) Attacks	173
6.6.2	Replay Attacks	174
6.7	Evaluation	175
6.7.1	Experimental Setup	177
6.7.2	Application Partitioning Performance	178
6.7.3	SL-Local Performance	178
6.7.4	Complete Performance Evaluation	179
6.7.5	Impact of Scalable SGX	181

6.8	Conclusion	181
7	SecureFS: A Secure File System for Intel SGX	183
7.1	Introduction	183
7.2	Background	185
7.2.1	A Primer on Intel SGX	185
7.2.2	Metadata organization of a File System	186
7.2.3	Attacks on File Systems	188
7.3	Related Work and Replay attacks	189
7.3.1	Graphene Protected Files	189
7.3.2	Nexus	190
7.4	Characterization	191
7.4.1	Experimental Setup	193
7.4.2	Access patterns	193
7.4.3	Distribution of File Access Calls over Time	194
7.4.4	Throughput	194
7.4.5	Key Insights	194
7.5	Design and Implementation	195
7.5.1	Design Goals	195
7.5.2	Threat Model	195
7.5.3	Overview of the Design	195
7.5.4	Organization	196
7.5.5	Storage on the Host File System	197
7.5.6	Key Structures of SecureFS	197

7.5.7	Metadata Organization	199
7.5.8	SecureFS-FAT	199
7.5.9	File System Operations	200
7.5.10	Preventing Replay Attacks	202
7.5.11	Concurrency and Consistency	202
7.5.12	SecureFS-inode	203
7.5.13	Implementation	203
7.6	Evaluation	203
7.6.1	Evaluation strategy	204
7.6.2	SecureFS-FAT vs SecureFS-inode	205
7.6.3	Throughput of the File System: Iozone	207
7.6.4	Comparison with GraphenePF: Regular applications	209
7.6.5	Comparison with Nexus	211
7.6.6	Sensitivity Analyses	212
7.6.7	Memory mapped file operations	213
7.7	Conclusion	214
8	OptMig: Secure and Efficient Migration of Large SGX Enclaves	215
8.1	Introduction	215
8.2	Background	218
8.2.1	Migration Methods	218
8.2.2	Intel Software Guard eXtensions	219
8.3	Related Work	220
8.4	Motivation and Characterization	221

8.4.1	Microbenchmark	221
8.4.2	Breakdown of the Total Downtime	221
8.4.3	Enclave Initialization & Down Time	222
8.4.4	Challenges in Adopting Live Migration for SGX	223
8.5	Design	225
8.5.1	Threat Model	225
8.5.2	Overview	225
8.5.3	OptMig library (OptMigLib)	227
8.5.4	High-level Operation	228
8.5.5	Save and Restore details	230
8.6	Fine-grained Access Tracking	230
8.6.1	OptMig Fault Tracker	231
8.6.2	OptMig Access Tracker	232
8.6.3	Discussion	234
8.7	Security Analysis	235
8.8	Evaluation	236
8.8.1	Experimental Setup	237
8.8.2	Fault Tracker v.s. Access Tracker	238
8.8.3	Improvement in the Total Downtime	238
8.8.4	End-to-End Performance (Single Migration)	238
8.8.5	Performance w.r.t. No Migration	240
8.8.6	Throughput Results (Key-Value Store)	241
8.8.7	Network Page Faults	242

8.8.8 Migration in Virtualized Environments	243
8.9 Conclusion	244
9 Conclusion and Future Work	245
Bibliography	249
List of Publications	289
Biography	291

List of Figures

1.1	An overview of the work done. We divide the contributions into three logical stages: ❶ setup ❷ securing a single application, and ❸ end-to-end solutions.	3
2.1	A control flow graph for OpenSSL. The vertices are functions in the application, and the edges between two vertices represent a direct function call.	12
2.2	Scope of this survey: We focus on hardware-assisted defense mechanisms for protecting the CFI of a program	13
2.3	Attacks on the control flow integrity or CFI of an application	16
2.4	Example of a buffer overflow attack on the code shown in Listing 2.3	17
2.5	Code injection, code reuse and control flow bending attacks	18
2.6	Two trusted execution environments or TEE solutions from Intel and ARM, respectively.	21
2.7	Last branch record or LBR	22
2.8	Collecting the branch trace on Intel processors. Adapted from [238]	23
2.9	Overview of Intel SGX.	24
2.10	Use of dedicated registers in a traditional pipeline [72].	25
2.11	A novel three-level taxonomy	30
2.12	Methods to generate a <i>contextual</i> control flow graph (or CCFG)	34

2.13 Working of Glamdring [233] using LLVM to split a process into secure and unsecure parts. Adapted from [233]	36
2.14 Working of OpaqueCFI [259]. Adapted from [259]	37
2.15 Working of PolyGlot [327]. Adapted from [327]	38
2.16 Storing of the keys in the page table in PolyGlot [327]. Adapted from [327] . . .	39
2.17 Architecture of HAFIX [130]. Adapted from [130])	41
2.18 Design of an heterogeneous-ISA platform [352]. Adapted from [352]	42
2.19 Architecture of Morpheus [152]. Adapted from [152]	46
2.20 Pointer displacement defense in Morpheus [152]. Adapted from [152]	46
2.21 Architecture of ASIST [114]. Adapted from [114]	47
2.22 Traditional approach for maintaining a shadow stack [127]. Adapted from [127]	49
2.23 Working of a parallel shadow stack [127]. Adapted from [127]	50
2.24 Return oriented programming or ROP attack on a stack [94]. Adapted from [94]	51
2.25 Spatial pointer corruption detected by using REST [326]. Adapted from [326] .	54
2.26 Architecture of HCIC [375]. Adapted from [375]	55
2.27 Working of ARM's pointer authentication code. The instruction <code>pacia</code> is used to generate the signature of the pointer and store it in the unused bits. The size of the signature depends on the system's configuration [230]. Adapted from [230]	56
2.28 Working of PACSafe [180]. Adapted from [180]	57
2.29 Architecture of Loki [374]. Adapted from [374]	58
2.30 Working of FineDIFT [109]. Adapted from [109]	59
2.31 Encoding and decoding in HW-CDI [226]. Adapted from [226]	61
2.32 Generation and storage of PACs [359]. Adapted from [359]	62
2.33 Overview of RetTag [359]. Adapted from [359]	63

2.34	Replacing indirect branches with direct branches [67]. Adapted from [67] . . .	66
2.35	Architecture of PathArmor [348]. Adapted from [348]	66
2.36	Architecture of BB-CFI [128]. Adapted from [128]	67
2.37	Tracing control flow in Griffin [154]. Adapted from [154]	69
2.38	Architecture of FlowGuard [237]. Adapted from [237]	70
2.39	Basic block boundaries [175]. Adapted from [175]	71
2.40	Cascaded updates in an inode-based design [215]. Adapted from [215]	72
2.41	Working of DATA [362]. Adapted from [362]	73
2.42	Overview of HeNet [110]. Adapted from [110]	73
2.43	Overview of Barnum [369]. Adapted from [369]	74
2.44	Overview of the Barun LSTM model [369]. Adapted from [369]	75
2.45	Working of DeepCheck [376]. Adapted from [376]	75
2.46	Working of NumChecker [353]. Adapted from [353]	76
2.47	Monitor systems in an HPC setting and validate it remotely [354]. Adapted from [354]	77
3.1	Figure showing the relation between the address space, the EPC, and the EPCM.	87
3.2	The performance overhead in SGX depends on the amount of secure memory (EPC) used by an application and the number of secure threads used (in a multi- threaded application). Furthermore, in a LibOS setting, the performance trends may change for different workloads on increase the use of EPC memory. . . .	91
3.3	Performance impact of SGX on applications in <i>Native</i> mode for different input sizes.	101
3.4	Performance impact of GrapheneSGX on workloads in SGXGauge.	102
3.5	Latency of core operations in Intel SGX.	105

3.6	Overheads for the workloads when executing in the <i>Native</i> mode w.r.t the <i>Vanilla</i> mode.	106
3.7	Figure showing the performance counter values for EPC page allocation, eviction, and load-back during the execution phase of B-Tree in the <i>Native</i> mode (N-) and <i>LibOS</i> mode (G-).	109
3.8	The I/O overhead with GrapheneSGX (S-G) and GrapheneSGX with protected files (S-P). Iozone: reading and writing 1 GB of data with 4 M blocks.	110
4.1	Normalization of the trend score of LLC misses for five workloads: PageRank, HashJoin, BFS, BTree, and OpenSSL)	120
4.2	Difference between coverage and spread. Suite WA has high coverage but a low spread. Suite WB has good coverage and spread.	122
4.3	Benchmark scores for three different settings: a) with all PMU counters b) only using LLC-related PMU counters, and c) only using TLB-related PMU counters.	126
4.4	Clustering in Nbench and SGXGauge.	127
4.5	Trend of LLC misses for Nbench and SPEC'17.	128
4.6	Figure showing the coverage of LMbench and SPEC'17 using the first two components of PCA [197].	128
5.1	Control flow of a license manager.	134
5.2	Call graph showing clusters, license check nodes, high-degree nodes, and bridge nodes for the <i>MySQL</i> benchmark	138
5.3	Visualization of the CFG across the pruning sub-phases for <i>MySQL</i> . (a) After clustering of the graph based on the nodes' properties, (b) after radius based pruning, and (c) after score and degree based pruning. The color of a node represents the cluster to which it belongs, and the black node represents the license check function.	140
5.4	Flowchart of the entire process	141
5.5	Overview of F-LaaS (shows the working when the license is valid)	143

5.6	Performance comparison of CFB vs SmartCFB (mean reduction in the number of functions is 22.3X)	149
6.1	Figure showing execution flows in the case of a valid license file and during a <i>control flow bending</i> or CFB [98] attack to bypass the authentication module.	152
6.2	Authentication logic of MySQL 5.7 and its corresponding assembly code. The license check function <code>do_auth_once()</code> is guarding the access to the protected region.	155
6.3	A high-level design of SecureLease.	164
6.4	Organization of leases. We index into different levels of the trees using the bits in the lease ID.	166
6.5	Procedure for committing a lease.	171
6.6	A high-level working of MySQL [262], two CFB attacks on it, and the partitioning done by SecureLease.	173
6.7	Figure showing the functions migrated by Glamdring and SecureLease for a representative benchmark OpenSSL.	178
6.8	Attestation performance for single and concurrent requests.	179
6.9	Performance overhead comparison for <i>F-LaaS</i> , Glamdring (Glam.), and SecureLease (SL) due to SGX, allocation requests with SL-Local (Local alloc.), and lease renewal.	180
7.1	Design of a FAT table-based metadata organization.	187
7.2	Design of an inode-based metadata organization.	188
7.3	Mounting a replay attack in Nexus [142].	190
7.4	Characterization of applications in terms of their access patterns, file system calls, amount of data accessed, and total files accessed.	192
7.5	Overview of SecureFS showing the key data structures, enclave boundary, chunk organization, and metadata entries.	196

7.6	Steps in opening and reading a file <code>/dir/car.jpg</code> in SecureFS-FAT.	201
7.7	Cumulative Distribution Function or CDF [261] of read and write latencies for Iozone: sequentially reading and writing 1 GB of data.	206
7.8	Total time taken to handle chunk cache misses and metadata cache misses. Iozone writing 1 GB of data. The x-axis shows the time.	206
7.9	SecureFS-FAT optimizes the metadata structure resulting in performance benefits.	206
7.10	Performance comparison for Iozone read and write operations.	207
7.11	CDF [261] of read and write latencies for Iozone while sequentially reading and writing 1 GB of data within Graphene.	208
7.12	dTLB events for G-PF, G-SF-F, and G-SF-I.	208
7.13	Comparison of the total time taken by G-SF-F and G-SF-I, w.r.t G-PF.	209
7.14	Total #asynchronous enclave exits (AEX).	210
7.15	Different <i>perf</i> [360] events for G-SF-F, normalized to G-PF.	210
7.16	Comparison of total time taken by SF-F and SF-I, with respect to Nexus as the baseline (not shown).	211
7.17	Sensitivity of SecureFS to different chunk sizes and chunk cache sizes.	212
8.1	OptMig ensures a constant total downtime while migrating an SGX-based application, irrespective of its size	216
8.2	Working of the pre-copy and post-copy migration mechanisms. The total downtime is independent of the memory footprint (mostly). Start indicates the start of migration, Pause and Resume indicate pausing and resuming the application, respectively.	217
8.3	For large enclaves, the total migration time is dominated by the enclave initialization time on the destination machine and the total transfer time.	221
8.4	A high-level overview of the working of OptMig.	224

8.5	Heap tracking in OptMig. OptMig overloads the <code>malloc</code> and <code>free</code> functions to keep track of all the allocated memory regions.	228
8.6	Generation of the required keys at the beginning of migration. The keys are sent synchronously to the destination. A key is generated for every heap page. .	228
8.7	Key components involved in saving the pages on the source machine and restoring them on the destination machine. Note that <code>save_vec</code> is allocated in the untrusted memory region so that CRIU can read it while servicing a network fault. However, <code>restore_vec</code> is allocated inside an enclave as there is no need on the destination machine for CRIU to read it.	228
8.8	Figure showing two different approaches for fine-grained access tracking using page faults (a) or automatically added access checks (b) and the comparison of the steps required by both approaches. NF: Network faults.	231
8.9	Total downtime using <i>MigSGX</i> , OptMig-V1, and OptMig-V2 across workloads. The main cause of downtime in <i>MigSGX</i> is the transfer time, but in OptMig-V1, it is the enclave <i>init</i> time.	240
8.10	Figure showing the drop in the throughput of Key-Value when migrating the application with three different techniques: <i>MigSGX</i> , OptMig-V1, and OptMig-V2.	242

List of Tables

1.1	Security requirements for data, code, and the execution flow.	2
1.2	Table comparing pure software, pure hardware, and hybrid-based defense methods.	3
2.1	Table showing a brief description of hardware features along with its use cases, overheads, and implementation details.	26
2.2	Related survey papers	27
2.3	Differences between the work done by Coppolino et al. [120] and this work. . .	28
2.4	The pre-execution class of defense methods.	33
2.5	Defense methods protecting key program components and structures.	44
2.6	The <i>system-based</i> class defense methods.	64
3.1	Conventions used in the chapter for discussion	94
3.2	Description of the workloads in SGXGauge along with the specific settings used in the chapter.	95
3.3	System configuration	99
3.4	Overhead in system-related events. Avg. value of EPC evictions is reported when compared with the <i>Vanilla</i> mode. The overhead refers to the performance overhead (run time).	100

3.5	Table showing the most important hardware performance counter that determines the performance of each workload (shown in bold). LLC refers to the last level cache.	108
4.1	Analysis of prior work in this area.	115
4.2	System configuration	123
4.3	Description of the different suites and workloads used in Perspector. All the benchmarks are executed with their standard input settings.	124
4.4	Description of the hardware counters.	124
5.1	Description of the hyper-parameters used in the <i>SmartCFB</i> algorithm	146
5.2	Table showing the final values of the hyper-parameters. The Percentile of the license check node is shown along with the score threshold ζ . The value of γ for degree-based filtering was set to 10 for all the benchmarks.	147
5.3	Table showing the rank of the degree of the license check function (percentile), and mean cohesion values of <i>O</i> -functions.	148
6.1	Lookup performance for different schemes.	168
6.2	Terminology	168
6.3	System configuration	175
6.4	Description of the workloads in SecureLease along with the specific settings used in the chapter.	175
6.5	Table showing the static and dynamic coverage for Glamdring (Glam.) and SecureLease (SLease), functions migrated by SecureLease, memory-related statistics for Glamdring and SecureLease, and the performance improvement in SecureLease.	176
6.6	Memory usage of SecureLease with and without eviction.	179
7.1	Description of the applications used in the chapter.	193

7.2	System configuration	204
7.3	Notations used in the chapter for discussion	204
7.4	Cache statistics for SecureFS-FAT and SecureFS-inode with Iozone while reading and writing 1 GB of data. used.	205
8.1	A summary of related work	220
8.2	Description of the workloads and the specific settings.	237
8.3	Table comparing the cost of serving a page fault in the restored enclave using Fault Tracker and Access Tracker. All times are in μs	237
8.4	Performance evaluation with <i>MigSGX</i> , OptMig-V1 (OM-v1), and OptMig-V2 (OM-v2). The performance improvement is for an end-to-end run.	239
8.5	Table showing end-to-end performance overhead due to: (i) additional memory checks only (no migration) (ii) with migration (<i>MigSGX</i> , OptMig-V1 (OM-v1), and OptMig-V2 (OM-v2)), vs no-migration.	241
8.6	Total number of network faults in different workloads with a <i>naive</i> placement of memory regions.	243
8.7	Performance evaluation of total downtime (DT) for OptMig on an academic cloud setup. Here, DT includes downtime due to sandbox transfer and SGX enclave transfer. Memory in MB. We perform <i>live</i> VM migration.	244