저작자표시-비영리-변경금지 2.0 대한민국

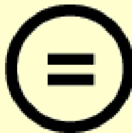
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

PassSSD: A Ransomware-proof SSD Using Fine-Grained I/O Whitelisting

PassSSD : 세분화 된 I/O 화이트리스트를 사용하는 랜섬웨어 방지 SSD

2021 년 8 월

서울대학교 대학원

컴퓨터 공학부

Nagmat Nazarov

PassSSD: A Ransomware-proof SSD Using Fine-Grained I/O Whitelisting

PassSSD : 세분화 된 I/O 화이트리스트를 사용하는 랜섬웨어 방지 SSD

지도교수 김 지 홍

이 논문을 공학석사 학위논문으로 제출함

2021 년 6 월

Seoul National University

Department of Computer Science and Engineering

Nagmat Nazarov

Nagmat Nazarov의 공학석사 학위논문을 인준함

2021 년 6 월

위 원 장	하 순 회
부위원장	김 지 홍
위 원	이 창 건

Abstract

PassSSD: A Ransomware-proof SSD Using Fine-Grained I/O Whitelisting

Nagmat Nazarov
Computer Science and Engineering
The Graduate School
Seoul National University

In recent years, Ransomware has been popular among cybersecurity experts because of the easy creation of new variants capable of bypassing anti-viruses and anti-malware. As Ransomware is targeting SSD (Solid State Drive) storage which is widely adopted in various emerging data-driven applications, modern storage systems are required to satisfy new requirements such as high security and protection. Therefore, it becomes crucial to develop new protection techniques that can properly address new challenges. In this thesis, we propose protection techniques that enable secure real-time flash storage systems without compromising performance and reliability. Our techniques are motivated by FTL (Flash Translation Layer) mechanism inside SSD and Hardware-based PrC (program context) register. Application whitelisting mitigates the ransomware attacks by specifying a list of pre-approved executable files allowed on a computer system, but this type of coarse-grained protection is vulnerable to control-flow hijacking attacks on safe application side. We propose PassSSD

- a fine-grained I/O whitelisting for a secure SSD using I/O (input/output) Program Contexts on RISCV architecture by modifying FlashBench [11] FTL.

We grant access to input/output requests based on whitelisted PrC's (program context) residing in FTL Superblock. In PassSSD, the PrC is extracted dynamically on *ext4_write_end()* function under inode.c on page granularity, where we attach PrC signature to every write syscalls and send (PrC,data) tuple to FlashBench FTL. On FlashBench FTL, the PrC monitor unit checks the incoming PrC from the Whitelist and allows the execution if PrC is whitelisted. By employing efficient protection on SSD, all unauthorized I/O requests are denied by the disk drive, logical page, and its physical page are separated by exploiting whitelisting technique. We implemented PassSSD on a FlashBench to verify the effectiveness of the proposed schemes. Our experimental results show that PassSSD can achieve the same performance level as a baseline scheme (FlashBench) with only 1% overhead bandwidth which is very negligible. [1]

Keywords: I/O Whitelisting, ransomware, FTL (Flash Translation Layer), program context, RISCV

Student Number: 2019-23358

[1]The author of this thesis is a Global Korea Scholarship scholar sponsored by the Korean Government

Contents

Abstract **i**

Contents **iv**

List of Tables **v**

List of Figures **vi**

Chapter 1 Introduction **1**

1.1 Motivation 2

1.2 Contributions 3

1.3 Thesis Structure 4

Chapter 2 Background **5**

2.1 Program Context 5

2.2 Whitelisting 6

2.3 Ransomware protection 7

2.3.1 SSD-Level Ransomware Defense 8

2.4 Limitations of existing Approaches 9

Chapter 3 PassSSD : Design and Implementation **11**

3.1 PassSSD Overview 12

3.2 Disk-level Access Control 13

3.3 Threat Model: Ransomware Attacks 15
3.3.1 Attacker Model 16
3.4 Use Cases 18

Chapter 4 Program Context **19**
4.1 Hardware-based PrC register 19
4.2 PrC(Program Context) Calculation 20
4.3 PrC Uniqueness 20
4.4 PrC Management Policy: 21
4.5 Page PrC Transfer and OS Support 22

Chapter 5 Evaluation and Experiments **24**
5.1 Experimental Methodology 24
5.2 Implementation Environment 25
5.3 Experimental Results 25

Chapter 6 Conclusions **27**
6.1 Summary 27
6.2 Future Work 28

Bibliography **29**

국문초록 **32**

List of Tables

Table 4.1 PrC Calculation mechanism 20

Table 5.1 SSD Configuration for experiment 25

Table 5.2 FTL overhead of PassSSD and FlashBench FTL 25

List of Figures

Figure 3.1 PassSSD Overall Architecture. 13
Figure 3.2 Control Flow hijacking attack on Application Whitelisting. 15
Figure 3.3 FlashBench Flash Translation Layer overview. 17

Figure 4.1 PrC Calculation. 21
Figure 4.2 Pass-SSD execution path. 22

Figure 5.1 The average IOPS comparison. 26

Chapter 1

Introduction

Ransomware is a type of malware from cryptovirology that threatens to publish the victim's data or perpetually block access to it unless a ransom is paid. While there exists simple ransomware which locks the system so that it is not difficult for a knowledgeable person to restore the system, more advanced malware uses a technique called cryptoviral extortion. It encrypts the victim's files, making them inaccessible, and demands a ransom payment to decrypt them [14, 18]. Ransomware has been popular among cybersecurity experts in recent years because of the easy creation of new variants capable of bypassing anti-viruses and anti-malware [16]. Apart from easy implementation, the high financial benefits of ransomware make it a profitable business, which motivates many cyber criminals to develop and distribute various ransomware programs [1].

There are two types of ransomware: locky and crypto. Locky ransomware locks the entire system from access by its user, but it is usually easy to resolve. However, crypto-ransomware uses encryption technology to lock selected files from user access; this is much more difficult to resolve and the damage caused may be irreversible. We will focus on Crypto ransomware in our thesis since it is a more popular type employed by cyber criminals [2]. By the end of 2021, it is estimated that a business will be targeted by a ransomware attack every 11

seconds, causing up to $20 billion in damage [9].

1.1 Motivation

The primary target of ransomware is data files created by a user through an application(e.g Microsoft Word, Adobe Reader). Application whitelisting is a good approach to mitigate ransomware attacks [10] where it specifies a list of pre-approved software applications or executable files allowed on a computer system. It does not require continuous scanning which reduces bandwidth, CPU, and memory. It is good protection against zero-day malware attacks [17]. Apart from pros, application whitelisting has the following cons :
1) An attacker may exploit safe applications (like Word processor, Adobe Reader) to execute malicious code.
2) Dynamic-link library (DLL) injection may place a copy of the malicious library into the system.
Typical ransomware goes through infection, persistence, removing backup copies, encryption, and notice. Specifically, we consider that ransomware may exhibit the following behaviors. As Application whitelisting (coarse-grained) is vulnerable to control-flow hijacking attacks [13] on safe applications, we propose doing fine-grained whitelisting based on PrC value of binaries where PrC uniquely represents an execution path of a program up to a write system call and value of PrC is calculated by summoning program counter (PC) values of each execution path of function calls.

Ransomware may access files through a regular file system or directly access the raw disk drive without going through the file system. Traditional access control mechanisms are implemented in the file system. If ransomware bypasses the file system, on conventional SSD's there is no more barrier in the disk drive.

Since the proposed access-control mechanism is implemented in the FlashBench FTL, it can prevent both direct access attacks and attacks through a regular file system.

1.2 Contributions

There are several methods to mitigate Ransomware attacks. The most well-known methods are Backup/Recovery based ransomware protection, Signature-based ransomware detection, System monitoring-based ransomware detection schemes, and Application whitelisting. As an example, Application Whitelisting tools such as Microsoft Applocker can choose applications that are approved for use. Therefore, applications that are not on the whitelist will get blocked from execution. Since *whitelisting* on application whitelisting is done by the host, it is easily exposable to various host-level attacks which may lead to control-flow hijacking attacks. To overcome these issues we designed and implemented ransomware-proof PassSSD using Fine-Grained I/O PrC by exploiting whitelisting inside SSD where whitelisting management is done by FTL.

This thesis describes i) PassSSD: A Ransomware-proof SSD using Fine-Grained I/O Whitelisting on FTL by exploiting program context register and ii) protection scheme inside SSD which protects from unauthorized I/O even when the OS is not trustworthy. We exploited the PrC register where each program context is identified by summoning program counter values of each execution path of function calls. The goal is to support fine-grained I/O whitelisting for a PassSSD. This thesis provides insight as to protect against Ransomware with low overhead by leveraging the PrC register on RISCV architecture. We also implemented proactive manner cybersecurity functions on FTL, which blacklists zero-day ransomware attacks if it keeps attacking the system. The main

contributions of this study are as follows: First, our experimental results show that PassSSD can protect the data against ransomware attacks, with minimal computation and space overheads for the PrC and whitelist management. Second, we implemented PrC fetch mechanism on Operating System.

Another challenge in designing PassSSD is the PrC register where we proposed hardware-based PrC signature calculation instead of software-based calculation. The latter has too much computation overhead and binaries are compiled with *fno-frame-pointer*, while our approach updates PrC on *jal/jalr* instruction and extended *CSR* registers to expose PrC to the OS. The key challenge on PassSSD is that we compute PrC on CPU (Trusted environment) without involving DRAM and SSD while achieving low space overhead since the computation is done on hardware.

1.3 Thesis Structure

The rest of the paper is organized as follows. In Chapter 2, we review FTL-based ransomware protection schemes and present a key motivation of our work. Chapter 3 describes the design and implementation of our PassSSD prototype and its protection mechanism. In Chapter 3 we explain program contexts and PrC fetch mechanism. We present our experimental environment and evaluation in Chapter 5, followed by a conclusion in Chapter 6.

Chapter 2

Background

The FTL (Flash Translation Layer) is a firmware made up of software and hardware which manages SSD operations like logical-to-physical address translation, garbage collection, wear leveling, block management, and error correction code. There are several approaches where the FTL mechanism has been exploited to mitigate ransomware attacks. To overcome backup overheads of host-level defense schemes, storage-level solutions have been proposed by Park et. al where the authors leveraged the out-of-place update property of NAND Flash to decrease back-up overheads [12]

2.1 Program Context

A key observation in the processor architecture is that program instructions (or their program counters) provide a highly effective means of recording the context of program behavior. Consequently, program-counter-based (PC-based) prediction techniques have been extensively studied and widely used in modern computer architectures. In PassSSD, we employ a write program context as such a higher-level classification unit for representing single I/O activity regardless of the type of I/O workloads. A program context (PrC) [6, 19], which uniquely represents an execution path of a program up to a write system call, is known to be effective in representing single I/O activities [7]. PrC value is calculated

by summoning return address values in the stack. By attaching PrC to I/O activities using program contexts during run-time on hardware, PassSSD can automate the whole process of PrC fetch from within the kernel with no extra overhead. We explain PrC in more detail in Chapter 4.

2.2 Whitelisting

A whitelist is a list of discrete entities, such as hosts, email addresses, network port numbers, runtime processes, or applications that are authorized to be present or active on a system according to a well defined baseline. Whitelists are primarily used as a form of access control: permitting activity corresponding to the whitelist and intended to stop the execution of malware and other unauthorized software. Unlike security technologies such as antivirus software, which blocks known bad activity and permits all other, whitelisting technologies are designed to permit known good activity and block all other.

An application whitelist is a list of applications and application components (libraries, configuration files, etc.) that are authorized to be present or active on a host according to a well-defined baseline[15]. On Application Whitelisting, whitelisting is managed by the *host* and used in an enterprise environment where granularity is coarse-grained and executed on the Application level. While on our approach whitelisting is managed by *FTL (Flash Translation Layer)* and used on SSD storage where granularity is fine-grained and executed on PrC(program context) level where we focus on write syscalls.

How does whitelisting software distinguish between unapproved and approved applications? The NIST (National Institute of Standards and Technology) guide breaks down the various attributes that can be used for this purpose:

a) The file name.

b) The file path.

c) The file size.

d) A digital signature by the software's publisher.

e) A cryptographic hash.

Choosing attributes is largely a matter of achieving the right balance of security, maintainability, and usability. In our approach, we chose PrC as the right attribute for whitelisting I/O. We also manage whitelisting inside SSD since managing whitelisting by the host (on Application whitelisting) is easily exposable to various host-level attacks.

Advantages of Whitelisting compared to Blacklisting: i) protects against zero-day malware attacks and doesn't need signature updates, ii) no continuous scanning required which in turn reduces bandwidth, CPU, and memory utilization, iii) only allowed applications, processes and executable are allowed to run and iv) unlicensed applications or software can be blocked which will eliminate copyright or license claims from the vendor. According to [5] whitelisting hasn't gained much popularity because of the misperception that it is very costly to implement. In our approach, we implemented fine-grained I/O whitelisting with 1% bandwidth overhead on SSD performance.

2.3 Ransomware protection

Microsoft's advice on tackling ransomware is that a tested reliable backup regime is the best way to mitigate the damage from a ransomware attack. While antivirus is still advocated, as we have seen, this may not be updated soon enough to block an attack. Microsoft also suggests AppLocker which blocks programs from running in common places. However, there is still a possibility

of new variants.

2.3.1 SSD-Level Ransomware Defense

Recently SSD-Level Ransomware defense is becoming popular. NAND flash memory does not support in-place updates, so SSDs handle host writes in an append-only manner, leaving obsolete file data unmodified and writing updates to different pages, and making the existing page invalid. Only SSD firmware, a flash translation layer (FTL) removes the invalid pages as blocks (groups of pages) and copies valid data on a block to a new block which is called GC (Garbage collection). Therefore, by preventing GC from erasing possibly attacked data, SSD-level solutions can back them up at no I/O cost. Furthermore, since these backup and removal processes are done inside SSDs, they can protect data from evasion attacks at the host level.

A ransomware tolerant SSD, FlashGuard was proposed by Huang et. al. which has a firmware-level recovery system that allows quick and effective recovery from encryption ransomware without relying on explicit backups [8]. FlashGuard leverages the observation that the existing SSD already performs out-of-place writes to mitigate the long erase latency of flash memories. Therefore, when a page is updated or deleted, the older copy of that page is anyway present in the SSD. FlashGuard slightly modifies the GC mechanism of the SSD to retain the copies of the data encrypted by ransomware and ensure effective data recovery.

SSD Insider [3] proposed a new approach to defending against ransomware inside NAND flash-based SSDs where features rely on observing the block I/O request headers only, and not the payload. For perfect and instant recovery, they proposed using the delayed deletion feature of SSDs, which is intrinsic to NAND flash. SSD-Insider addressed the limitations of FlashGuard by leveraging

a ransomware detection algorithm on the storage side. By monitoring unique I/O footprints of ransomware, it detects whether a host is under a ransomware attack or not within 10 seconds. This early detection enables SSD-Insider not to maintain backup data for a long time, which mitigates the high space overhead problem. SSD-Insider, however, could be easily compromised by ransomware variants that intentionally issue modified I/O patterns that evade the SSD-Insider's detection algorithm.

Ransomblocker [12] was proposed to defend against all possible ransomware attacks by delaying every data deletion until no attack is guaranteed. To reduce storage overheads of the delayed deletion, Ransomblocker employs a time-out-based backup policy. Based on the fact that ransomware must store an encrypted version of target files, early deletions of obsolete data are allowed if no encrypted write was detected for a short interval.

2.4 Limitations of existing Approaches

FlashGuard [8] keeps all invalid pages for 2-4 weeks if the pages have been read before invalidation. This is based on the fact that ransomware has to read and encrypt victim files. While it may be the most thorough way to protect user files against ransomware attacks, GC and space overheads cannot be avoided. By analyzing block I/O traces collected from enterprise systems, FlashGuard's authors conclude that only a few files are removed after being read. However, many counterexamples can be found in various scenarios. For a simple example, suppose that a user downloads a film and removes it after watching if he doesn't like it. With FlashGuard, this file will not be removed from an SSD for a very long time, unnecessarily occupying huge space.

SSD-Insider[4] maintains suspicious data for a short period. Thus, SSD-

Insider does not waste lots of free space to keep suspicious data and does not seriously suffer from GC overheads. The detection algorithm of SSD-Insider, however, can be easily compromised by ransomware variants. For example, SSD-Insider guesses that there may be ransomware attacks if a sharp increase of overwrite requests from a host is observed for a short time interval. If ransomware intentionally slows down its encryption speed, SSD-Insider may fail to detect it, allowing GC to permanently erase victim data.

Chapter 3

PassSSD : Design and Implementation

We first introduce the overview of PassSSD in Section 3.1. Then we describe the Disk-level Access Control in Section 3.2. Finally, we present the threat model and use cases in Section 3.3 and Section 3.4, respectively.

The proposed approach is the implementation of a ransomware-proof SSD Using Fine-Grained I/O whitelisting. We integrated an access control mechanism inside the SSD disk drive by modifying FlashBench FTL. The access-control mechanism allows access to files under protection only for authorized (Whitelisted) program context I/O. This access-control mechanism specifically targets preventing ransomware. This access-control mechanism can be implemented in the file system. But, compared to file system implementation, disk-level implementation has the following advantages.

Security: Since a disk drive is a separate system from a host, it is not easy to compromise both host and disk drive at the same time. Especially, if the disk drive does not allow firmware update by the host, it is very difficult to compromise the disk drive unless the disk drive is physically accessible.

Compatibility: The implementation of the disk-level access-control mechanism is independent of a file system. To support PassSSD, a small additional

kernel modifications need to be done to the file system.

We have designed and implemented a prototype ransomware-proof PassSSD, using fine-grained I/O Program Context, where we support Kernel PrC fetch by exploiting Hardware-based PrC calculation. Fine-Grained I/O Whitelisting is supported on FlashBench FTL. Apart from that, we support proactive manner cybersecurity functions, where our prototype blocks unauthorized access. If the adversary keeps requesting the write(), PassSSD adds that application to the blacklist where it can be used to avoid zero-day attacks.

3.1 PassSSD Overview

In this section, we will briefly discuss our key design principles. We envision PassSSD to be ransomware-proof protection to block ransomware attacks. An overview of a PassSSD is shown in Figure 3.1 presents a bottom-up description of the system for each component necessarily implemented at every level of OS stack and device drive. On our prototype PassSSD, the PrC is extracted dynamically on *ext4_write_end()* function under inode.c on page granularity. PassSSD attaches PrC signature to every write syscalls on page granularity to each LPA where our page size is 4KB and sends (PrC, data) tuple to FlashBench FTL. On FlashBench FTL, the PrC monitor unit checks the incoming PrC from the Whitelist and attaches designated PPA if PrC is whitelisted or discards if not.

Specifically, we have implemented a PrC management framework in OS kernel and the FTL extension for disk-level access control management on the SSD with the following main design goals: (i) protection from unauthorized I/O even when the OS is not trustworthy and (ii) lightweight PrC management in OS kernel.

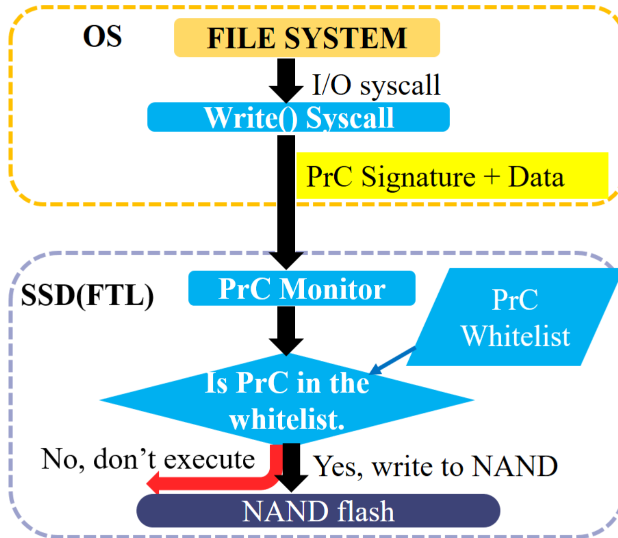


Figure 3.1 PassSSD Overall Architecture.

3.2 Disk-level Access Control

As we have mentioned, there is intelligent Ransomware that can evade system-level protection, that's why we need a more robust defense mechanism. Ransomware can directly access the disk without file system intervention since Ransomware can encrypt the disk. That's why we implemented the protection scheme inside the SSD to protect from unauthorized I/O even when the OS is not trustworthy. It divides OS (Operating System) and FTL to support fine-grained I/O to guarantee ransomware protection. FlashBench FTL, which is a firmware in the SSD, manages PrC (program context) per page. To authorize a request the OS needs to put a request along with the PrC. The PrC is sent together with the data on the file system layer until the write reaches the FTL. On FTL, requested pages are allowed to execute if PrC is whitelisted.

In this thesis, we implemented PassSSD as a prototype on RISCV archi-

tecture where we support Hardware-based PrC calculation. When the device receives a write command, the authentication is granted as follows. When SSD receives a write request, the SSD firmware first reads the PrC from the FlashBench FTL and compares it with the PrC sent by the host. If the value of a PrC is different, it sends an error message back to the host without performing the write operation because it is not authenticated. If the PrC value is the same as that in the FlashBench FTL, write operation is allowed.

In Linux OS, it is possible to write directly from block devices without the file system intervention. The attacker can directly access the disk block without knowing the contents and can encrypt the data (e.g. read sector 0 - 100 and encrypt the data). By developing a protection mechanism inside Flashbench SSD, we overcome this problem.

FlashBench-FTL: PassSSD extends the FlashBench FTL, which is firmware in the SSD, manages the PrC per page. To authorize a request, an application does not need to put a request along with the PrC. It is done automatically on a write system call. On each write syscall, PrC is attached along with the page on the file system layer until it is executed on FTL.

Fine-Grained input-output whitelisting: Data blocks must be given access based on whitelisted PrC value to grant access to the data page. This access control must be implemented within the disk drive. All data blocks need to be protected based on PrC whitelisting. The data pages must be protected by the I/O whitelisting technique according to the PrC value on binary files. It is also necessary to minimize the performance and space overhead of managing the PrC per page in the disk drive. In addition, the cost of keeping this information persistent should be minimized.

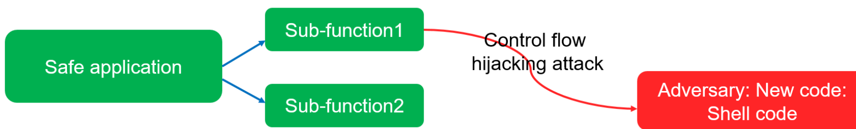


Figure 3.2 Control Flow hijacking attack on Application Whitelisting.

Flush daemon: *Sync* is a standard system call in the Unix OS, which commits all data in the kernel filesystem to non-volatile storage buffers, data that has been scheduled for writing via low-level I/O system calls. Higher-level I/O layers such as stdio may maintain separate buffers of their own. Some Unix systems run a kind of *flush daemon*, which calls the *sync* function regularly. Buffers are also flushed when file systems are unmounted or remounted read-only. In our implementation, the flush daemon writes back the page cache to SSD along with the PrC.

Inode protection: In PassSSD, both the user data and the metadata of the file are protected by the PrC. Whenever there is a write request we first update the data and then we access the inode table, inode bitmap, and data bitmap to make the updates. We encountered a metadata inconsistency problem when we were attaching PrC on the *ext4_da_write_end* function. To solve this problem, we changed the PrC attach function to the *ext4_write_end* function where we used *no_delay_allocation*.

3.3 Threat Model: Ransomware Attacks

The primary target of ransomware is data files created by a user through an application (e.g Microsoft Word, Adobe Reader). Application whitelisting is a good approach to mitigate ransomware attacks. It is good protection against

zero-day malware attacks. Apart from pros, application whitelisting has the following cons :
1) An attacker may exploit safe applications (like Word processor, Adobe Reader) to execute malicious code.
2) Dynamic-link library (DLL) Injection may place a copy of the malicious library into the system. Typical ransomware goes through infection, persistence, removing backup copies, encryption, and notice. Specifically, we consider that ransomware may exhibit the following behaviors. As Application whitelisting (coarse-grained) is vulnerable to control-flow hijacking attacks (figure 3.2) on safe applications, we propose doing fine-grained whitelisting based on the program context (PrC) value of binaries.

3.3.1 Attacker Model

Ransomware may be a user-level application with a root privilege. Ransomware can acquire a root privilege by exploiting vulnerabilities in applications or operating system. Once it has a root privilege, it can access the files of other users. The proposed technique can defend files against this type of ransomware too.

Ransomware may hijack system calls, but cannot access kernel data structures in file system layers. System call hijacking is one of the most popular techniques to implement a rootkit (kernel-level malware). A system call may be replaced by a malicious one. However, it is assumed that kernel data structures cannot tamper because it requires recompilation of the kernel, which is more challenging than hijacking.

We modified FlashBench FTL on our prototype PassSSD. As in figure 3.3, we separated FTL from OS and implemented PrC manager unit inside FTL which manages access control on disk drive using whitelisting.

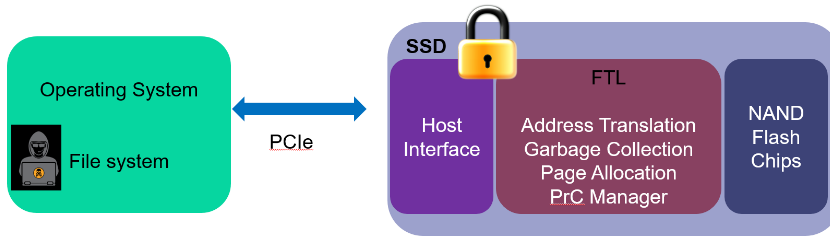


Figure 3.3 FlashBench Flash Translation Layer overview.

If the attacker exploits root/supervisor privileges and turns off any anti-virus daemons or system monitoring tools: This attack exploits a common remote code execution vulnerability, which gives an attacker a remote shell on the device. This attack can be considered as the entry point for other subsequent attacks. PassSSD can prevent unauthorized data modifications even if the kernel is not trustworthy, since we exploit FlashBench FTL which is decoupled from the OS.

The attacker can directly write from the disk: In Linux OS, it is possible to write directly from block devices without the file system intervention. The attacker can directly access the disk block without knowing the contents and can encrypt the data. It is the reason why we need to develop a more robust protection mechanism inside the SSD.

How do we guarantee that PassSSD is safe under the assumed attack model? We need to consider if there is any chance that malware's PrC colliding with the authorized program's PrC. We need to prove that each PrC is a unique signature, and the chance of having the same PrC among different programs is extremely low. The attacker might be able to create a binary that

has the same PrC as the authorized program, but assuming that the attacker knows one authorized PrC "A", generating a program that has the same PrC is not simple, since the PrC value depends on the library the program uses and plus the program binary (this means the attacker needs to analyze the library of the system).

3.4 Use Cases

Our PassSSD prototype can be used on Bitcoin Hot Wallet (also known as "Online Wallets") security-sensitive application for storing coins. Hot wallets are wallets that run on internet-connected devices like computers, phones, or tablets. This can create vulnerability because these wallets generate the private keys to your coins on these internet-connected devices. While a hot wallet can be very convenient to access and make transactions with your assets quickly, they also lack security. Apart from Hot Wallets, PassSSD can be used to back up the entire bitcoin wallet early and often. In case of a computer failure, a history of regular backups may be the only way to recover the currency in the digital wallet.

Chapter 4

Program Context

A key observation in the processor architecture is that program instructions (or their program counters) provide a highly effective means of recording the context of program behavior. Consequently, program-counter-based (PC-based) prediction techniques have been extensively studied and widely used in modern computer architectures. In PassSSD, we employ a write program context as such a higher-level classification unit for representing I/O activity regardless of the type of I/O workloads. A program context (PrC) [6, 19], which uniquely represents an execution path of a program up to a write system call, is known to be effective in representing single I/O activities [7].

In this paper, we argue that a program context can be used to build the right abstraction for input/output activities on different binary programs. Here, a PrC represents an execution path of an application that invokes input/output related system call function write(). We could extract PrC using various methods, but the most common approach is to represent each PrC with its signature.

4.1 Hardware-based PrC register

We create a new 64-bit CSR (Control and Status Register) register PC-CSR (program context-CSR). The address space of the actual program is 48bit. As you see in Table 4.1 Hardware automatically updates the register value ac-

Table 4.1 PrC Calculation mechanism

	JAL \$(ra),\$(offset)	Function Call
Jump Instruction (JAL,JALR)	JALR \$(zero),\$(ra)	Function return
	JALR \$(ra),\$(dest register)	Function Call

cording to the Program Context calculation rule. Only the lower N bits of the address can cause almost no overflow in the current Program Context addition/subtraction method. The register can be read and modified with the "csr" assembly instruction only in supervisor mode. On Function call, *PC = PC + (Current Program Counter) + 4* and on function return *PC = PC - (return address).*

4.2 PrC(Program Context) Calculation

Program context (PrC) value is calculated by summoning return address values in the stack. As you see in figure 4.1 the PrC value in function *boo* will be the sum of the return address of foo, boo, and bar which makes *PrC = 1004+2004+3204*. By attaching the PrC to I/O activities using PrC register during run-time on hardware, PassSSD can automate the whole process of PrC fetch from within the kernel with no extra overhead.

4.3 PrC Uniqueness

We executed more than 100 binaries with write syscalls and didn't encounter a single unique PrC coming from different paths. Even if PrC calculation is composed of addition/subtraction of return address, by changing orders of 10 functions each with write syscall it is extremely difficult to get the same PrC

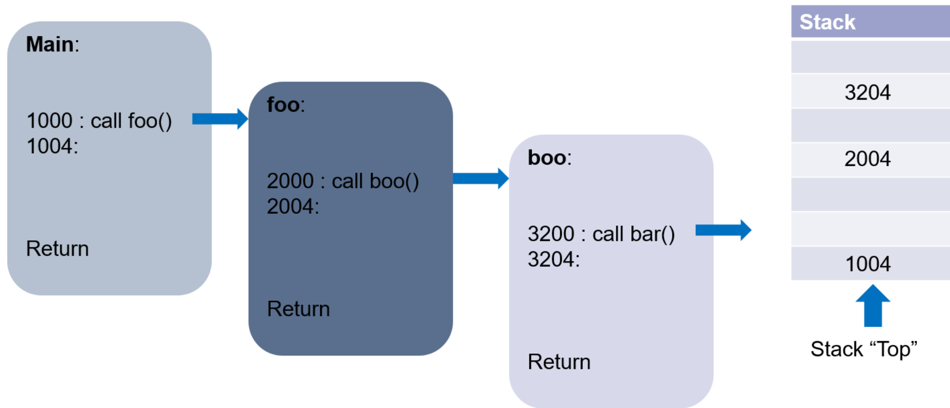


Figure 4.1 PrC Calculation.

with the permutation on that functions.

4.4 PrC Management Policy:

PrC Whitelisting management policy is very important in our prototype. In our implementation, we have implemented a supervisor application where we safely add/delete PrC and write the latest whitelisted PrC's to the superblock on the application layer where only the authorized "Supervisor" Program can read or update the whitelist table. We can achieve this by only allowing the supervisor program's unique PrC to access the whitelist table that resides in the superblock. This means even authorized whitelisted programs cannot modify the PrC whitelist. In our Supervisor application, we load the whitelist table on initialization from the Superblock. Then insert, delete and print procedures are executed on the loaded array. Finally, we flush back the updated whitelist table to Superblock on exit from the Supervisor program. The reason why we have implemented a whitelist on SuperBlock is that the file system automatically

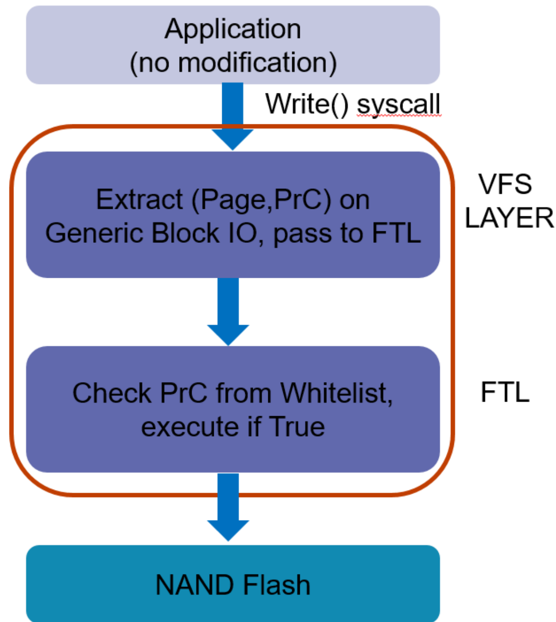


Figure 4.2 Pass-SSD execution path.

takes the Backups of SuperBlock. This way we can guarantee the persistence of our whitelist.

Minimal OS support for Pass-SSD: Figure 4.2 shows the execution path of the PassSSD prototype. We don't need to make any changes to the Application Layer. We attach PrC to each page on the file system layer until the write is executed on FlashBench FTL. To differ file I/O requests, the OS must be able to assign the PrC only to the page requests of file I/O operations.

4.5 Page PrC Transfer and OS Support

To perform PrC authentication per page request on a disk drive, it is essential to transfer the PrC to the disk drive. The PrC corresponding to the LPA requested

by the host must be correctly registered in the FlashBench FTL of the PassSSD. Importantly, the PrC must be sent synchronously with the LPA page request. After the page request is completed, the PrC must be deleted from the OS kernel to minimize space overhead.

Chapter 5

Evaluation and Experiments

In this chapter, we provide comprehensive experiments to evaluate the effectiveness of the PassSSD. We demonstrate how PassSSD detects and stops ransomware. We will also discuss how PrC is important in detecting ransomware in different ransomware attacks.

5.1 Experimental Methodology

We evaluate PassSSD on a modified RISCV freedom-u-sdk and FlashBench FTL simulator [11]. We simulated a 1GB capacity flash-based SSD with a 4 KB page size and around 26200 blocks as in table 5.1. We modified the existing structure of FlashBench [11] FTL to support PrC based whitelisting approach. For experiments, we used 4 different variants of ransomware that behave differently in terms of attack patterns. We performed a step-by-step analysis of the usefulness of the PassSSD for a ransomware attack. Then, introduce the FTL overhead of PassSSD concerning FlashBench FTL excluding Kernel overhead since Kernel overhead is negligible.

Table 5.1 SSD Configuration for experiment

SSD Capacity	1024 MB	Pages per Block	26200	Page Size	4KB

Table 5.2 FTL overhead of PassSSD and FlashBench FTL

Name	Instruction count
PassSSD	1309 million
FlashBench FTL	1284 million

5.2 Implementation Environment

To prototype the PassSSD, we modified 150 lines of C code in the Linux VFS and the generic block I/O layers, 100 lines of C code in the FlashBench FTL [11].

Testbed: All experiments were performed on a simulator with a core design that incorporates a 5-stage single-issue in-order pipeline. It implements the 64-bit RV64GC scalar RISCV ISA. The operating system is Linux kernel 4.15 with 4 processors.

5.3 Experimental Results

In our experiments, we tested our prototype PassSSD against 4 different ransomware attacks where PassSSD was successful to overcome the attack. The important thing in our experiments is that all 4 ransomware were attacking through unique PrC, which shows how PrC abstraction is the right approach to detect input/outputs. The adversary tries to attack the safe binaries and overwrite the data inside executable binaries into the attacker's code.

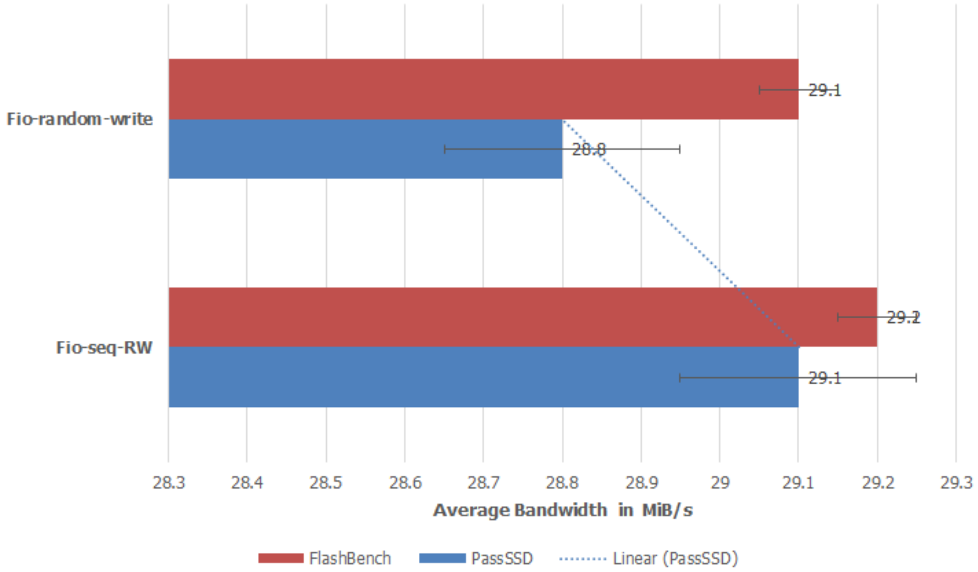


Figure 5.1 The average IOPS comparison.

Apart from the ransomware protection, we measured IOPS (Input output operations per second) overhead (any combination of excess or indirect computation time, memory, bandwidth, or other resources that are required to perform a specific task) of PassSSD where we modified FlashBench FTL with baseline FlashBench(the one before modification). We excluded kernel overhead while transferring PrC from Operating System since it's negligible in comparison with FTL overhead. In table 5.2 you can see the average IOPS of our PassSSD FTL and FlashBench FTL on *fio-random-write* and *fio-seq-read-write* workloads. We tested our prototype with 100 whitelisted PrC's and 10 different binaries. From Figure 5.1, PassSSD achieves ransomware-proof property on our prototype with less than 1 % IOPS overhead which is not a big burden for our prototype.

Chapter 6

Conclusions

6.1 Summary

We have presented a novel Ransomware-proof PassSSD using Fine-Grained Input/Output Whitelisting with FlashBench FTL, to protect sensitive data residing on SSD without degrading the SSD performance. By employing efficient protection on SSD all unauthorized I/O requests are denied by the disk drive, logical page, and its physical page are separated by exploiting whitelisting technique. Furthermore, PassSSD exploited hardware-based PrC(program context) calculation, which significantly decreased the calculation overhead and space overhead. We implemented PassSSD on a FlashBench to verify the effectiveness of the proposed schemes. Our experimental results show that PassSSD can achieve the same performance level as a baseline scheme with only 1% IOPS overhead by SSD performance which is very negligible.

Lastly, we do note that our method shows high performance while fetching PrC on Kernel and exploiting Hardware-based PrC calculation. Similar approaches have been implemented using software stack but it had high computation and space overhead, which is why overall performance may have degraded.

6.2 Future Work

The current version of PassSSD can be further improved in several directions. For example, if we can utilize the control flow integrity while extracting PrC on applications we can dynamically allocate whitelisted PrC's. The metadata of the file structure can be further improved to robust metadata inconsistency. Also thinking of extra security enhancements, when the program is loaded, the initial PrC is set as the program binary hash value instead of zero, since generating a program that has the "same hash value" + "same PrC" is extremely difficult.

Bibliography

[1] AhnLab. Asec report: Ransomware paper on 2020. In *Ahn Lab report*, 2020. Accessed: 2021-04-22.

[2] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security*, 74:144–166, 2018.

[3] SungHa Baek, Youngdon Jung, Aziz Mohaisen, Sungjin Lee, and DaeHun Nyang. Ssd-insider: Internal defense of solid-state drive against ransomware with perfect data recovery. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 875–884, 2018. doi: 10.1109/ICDCS.2018.00089.

[4] SungHa Baek, Youngdon Jung, Aziz Mohaisen, Sungjin Lee, and DaeHun Nyang. Ssd-insider: Internal defense of solid-state drive against ransomware with perfect data recovery. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 875–884. IEEE, 2018.

[5] Aaron Beuhring and Kyle Salous. Beyond blacklisting: Cyberdefense in the era of advanced persistent threats. *IEEE Security & Privacy*, 12(5): 90–93, 2014.

[6] Chris Gniady, Ali Raza Butt, and Y Charlie Hu. Program-counter-based pattern classification in buffer caching. In *Osdi*, volume 4, page 27, 2004.

[7] Keonsoo Ha and Jihong Kim. A program context-aware data separation technique for reducing garbage collection overhead in nand flash memory. *Proc. 7th IEEE SNAPI*, 2011.

[8] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K Qureshi. Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2231–2244, 2017.

[9] KasperskyLab. Top ransomware 2020, causing 20billion in damage. In *Kaspersky report*, 2020.

[10] DaeYoub Kim and Jihoon Lee. Blacklist vs. whitelist-based ransomware solutions. *IEEE Consumer Electronics Magazine*, 9(3):22–28, 2020. doi: 10.1109/MCE.2019.2956192.

[11] Sungjin Lee, Jisung Park, and Jihong Kim. Flashbench: A workbench for a rapid development of flash-based storage devices. In *2012 23rd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 163–169. IEEE, 2012.

[12] Jisung Park, Youngdon Jung, Jonghoon Won, Minji Kang, Sungjin Lee, and Jihong Kim. Ransomblocker: a low-overhead ransomware-proof ssd. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[13] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 303–312. IEEE, 2016.

[14] Jack Schofield. How can i remove a ransomware infection? *The Guardian*, 2016.

[15] Adam Sedgewick, Murugiah Souppaya, and Karen Scarfone. Guide to application whitelisting. *NIST Special Publication*, 800(167), 2015.

[16] PEP Torres and SG Yoo. Detecting and neutralizing encrypting ransomware attacks by using machine-learning techniques: A literature review. *Int. J. Appl. Eng. Res*, 12(18):7902–7911, 2017.

[17] Hasan Turaev, Pavol Zavarsky, and Bobby Swar. Prevention of ransomware execution in enterprise environment on windows os: Assessment of application whitelisting solutions. In *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, pages 110–118, 2018. doi: 10.1109/ICDIS.2018.00024.

[18] Adam Young and Moti Yung. Cryptovirology: Extortion-based security threats and countermeasures. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 129–140. IEEE, 1996.

[19] Feng Zhou, J Robert von Behren, and Eric A Brewer. Amp: Program context specific buffer caching. In *USENIX Annual Technical Conference, General Track*, pages 371–374, 2005.

국문초록

최근 몇 년간 랜섬웨어는 안티바이러스 및 악성 프로그램 탐지 우회 기능을 갖춘 다수의 새로운 변종이 나타나면서 사이버 보안 전문가들 사이에서 인기를 끌고 있다. 많은 수의 랜섬웨어는 데이터 중심 애플리케이션에 널리 채택되고 있는 SSD 저장 장치를 공략하고 있지만, 현재 가장 널리 사용되고 있는 랜섬웨어 보호 방식들은 성능 및 신뢰성에 한계가 있다. 따라서 본 논문에서는 성능과 신뢰성을 훼손하지 않고 실시간으로 안전한 SSD 저장 장치 PassSSD를 제안한다.

PassSSD는 프로그램의 I/O 수행 맥락인 I/O 프로그램 컨텍스트 서명을 활용한다. I/O 프로그램 컨텍스트 서명은 응용이 저장 장치에 접근할 때의 고유한 서명 정보로 본 연구에서 개발한 하드웨어에 의하여 계산이 된다. 이러한 서명 정보를 SSD 내 FTL (Flash Translation Layer)에 [11] 전달함으로써, 응용이 수행하고 있는 I/O에 대해서 스토리지가 자체적으로 접근 권한을 판단하게 된다.

PassSSD는 FTL 내에서 화이트리스팅 기술을 활용한다. 화이트리스팅은 관리자가 사전에 특정 서비스 또는 액세스에 대한 접근 권한을 사전 승인된 프로그램에 대해서만 허용하는 정책으로, PassSSD는 사전에 등록된 I/O 프로그램 컨텍스트에 대해서만 디스크 접근을 허가하게 된다. 즉 PassSSD는 응용이 요청한 I/O 에 대한 있는 I/O 프로그램 컨텍스트 서명을 확인하고, 해당 서명이 화이트리스트에 있다면 저장 장치 내 데이터 접근을 허용하며, 없다면 이를 거부한다. 이로써 랜섬웨어 와 같은 무단 I/O 요청은 거부되고 저장 장치 내 데이터를 보호할 수 있다.

실험 결과 PassSSD의 FTL은 전체 수행 명령어 대비 1.9%의 적은 추가 명령어 수행만으로 성능 및 데이터 신뢰성을 보장할 수 있음을 확인하였다.

주요어: I/O 화이트 리스팅, 랜섬웨어, FTL, 프로그램 컨텍스트, RISCV
학번: 2019-23358