SSD GUARD: 높은 권한의 데이터 변조 공격으로부터 파일 보호

안진우[1], 이중희[2], 김영재[1]
[1]서강대학교 컴퓨터공학과, [2]고려대학교 정보보호학과
{jinu37, youkim}@sogang.ac.kr, j_lee@korea.ac.kr

SSD GUARD: File Protection Against High Priviledged Data Tampering Attacks

Jinwoo Ahn[1], Junghee Lee[2], Youngjae Kim[1]
[1]Dept. of Computer Science and Engineering, Sogang University, Seoul, South Korea
[2]School of Security, Korea University, Seoul, Republic of Korea

요 약

데이터의 중요성이 날로 증가함에 따라, 데이터를 파괴하는 멀웨어의 위협이 커져가고 있다. 특히, 멀웨어가 사용자의 컴퓨터에 침입하여 커널 권한을 획득하면, 운영체제가 제공하는 모든 보안 수단이 무효화 될 수 있다. 본 논문에서는 Intel SGX를 활용하여 스토리지 레벨에서 커널 권한의 멀웨어로부터 데이터 변조를 방어하는 시스템(SSD GUARD)를 제안한다. SSD GUARD는 SSD 펌웨어 안에 디바이스 레벨 파일 시스템과 인증 모듈을 구현한다. 이를 통해 TEE 기반의 애플리케이션이 취약한 운영체제의 파일시스템을 우회하여, 안전한 파일 연산을 가능하게 한다. SSD GUARD의 시스템을 증명하기 위해 우리는 Intel SGX가 제공하는 IPFS와 Jasmine OpenSSD 플랫폼 내부의 펌웨어 코드를 수정하여 end-to-end 프로토타입을 구현하였다. 실험 결과, SSD GUARD는 쓰기와 읽기 성능이 베이스라인에 비해 최대 33%, 2% 저하되었지만, 파일의 데이터 변조 방어라는 새로운 보안성을 제공한다.

1 서 론

멀웨어가 사용자의 컴퓨터에 침입하여 높은 권한을 획득하면, 컴퓨터에 저장된 모든 데이터는 훼손될 위협에 노출된다. 최근들어 수많은 사람들이 랜섬웨어 [1] 또는 와이퍼 [2]의 데이터 변조 공격으로 큰 피해를 입었다. 랜섬웨어는 사용자의 파일을 인질로 잡아 암호화하여, 이를 복호화하기 위한 금전을 요구하는 멀웨어이다. 반면 와이퍼는 오직 사용자의 파일을 파괴하는 것을 목적으로 한다. 이러한 멀웨어가 커널 권한을 획득한다면, 운영체제가 제공하는 기존의 보안 수단이 무효화될 위험이 있다 [3, 4]. 예를 들어, Shamoon2 [3]은 루트킷 권한을 탈취하고, 커널 드라이버를 이용하여 OS의 파일 시스템이 제공하는 보안을 우회하고 파일을 파괴한다.

Intel Software Guard eXtension (SGX) [5]가 개발되면서, 사용자 애플리케이션은 신뢰 실행 환경 (TEE)을 제공받을 수 있게 되었다. Intel SGX는 프로세서가 직접 애플리케이션의 기밀성과 무결성을 보증하기 때문에, 멀웨어가 커널 특권을 획득할지라도, 애플리케이션을 읽을 수도, 변조할 수도 없다. Intel은 여기서 더 나아가, SGX 기반의 애플리케이션이 생성한 파일을 안전하게 보관하기 위한 Intel Protected File System (IPFS) [6]를 개발하였다. IPFS는 애플리케이션 안에 고립된 키를 이용하여 데이터를 스토리지에 쓸 때 암호화 및 서명을 하고, 읽을 때 복호화와 증명을 한다. 따라서 커널 권한의 멀웨어조차 애플리케이션이 생성한 파일을 읽을 수 없다. 뿐만 아니라, 멀웨어가 파일을 변조하면 애플리케이션은 파일을 읽을 때 증명하여 변조를 바로 알아차릴 수 있다. 하지만, IPFS는 멀웨어의 변조 공격으로부터 데이터를 보호하기에는 역부족이다. IPFS는 파일을 읽을 때 변조를 증명할 뿐, 파일 데이터를 복구할 수 없기 때문이다. 이는 랜섬웨어나 와이퍼가 공격할 경우, 소 잃고 외양간 고치는 격이다.

본 논문에서는 로컬 환경에서 SGX 기반의 애플리케이션이 생성한 파일을 커널 권한의 멀웨어의 변조 공격으로부터 보호하는 스토리지(SSD GUARD)를 제안한다. SSD GUARD는 SSD 펌웨어를 하나의 TEE로 간주한다. SSD 펌웨어는 SSD에 내재된 cpu와 메모리로 in-device computation이 가능할 뿐 아니라, 디지털 인증 및 시큐어 부팅을 제공하므로 오직 제조사만이 업데이트할 수 있기 때문이다. SSD GUARD에서 애플리케이션과 스토리지는 TEE로 기능하며, 따라서 인증을 통해 서로 안전하게 커뮤니케이션을 한다.

하지만, 애플리케이션이 생성한 파일을 스토리지에 안전히 저장하는 것은 쉬운 문제가 아니다. 멀웨어는 커널에 있는 네이티브 파일 시스템의 기능을 훼손할 수 있기 때문이다. 파일 시스템은 애플리케이션이 생성한 파일 시맨틱을 디스크 블록으로 번역하는 중요한 역할을 담당한다. 만일 멀웨어가 파일 시스템의 아이노드를 암호화하면, 번역 정보가 사라지므로 애플리케이션은 다시는 파일을 읽을 수 없다.

파일 시스템 훼손 가능성을 차단하기 위해, SSD GUARD는 스토리지 내부에 보호 파티션을 할당하여, 이를 관리하는 가벼운 디바이스 레벨 파일 시스템을 설계한다. 그리고 애플리케이션의 파일 연산 요청을 네이티브 파일시스템을 우회하여, 스토리지에게 직접 전송한다. 네이티브 파일 시스템 대신 안전하게 스토리지에 고립된 디바이스 레벨 파일시스템이 파일 연산 요청을 수행한다.

하지만, SSD의 메모리 공간은 컴퓨터의 메모리 공간보다 제한되어 있기 때문에, 파일 시스템을 올리면 공간 오버헤드가 지나치게 커지는 문제가 발생할 수 있다. 이를 해결하기 위해 GUARD FTL을 제안한다. 기존에는 파일이 SSD에 저장되기 위해 파일 시스템에서 파일 오프셋이 논리 블록 주소로 번역되고, SSD 펌웨어의 Flash Translation Layer (FTL) 로부터 실제 물리 페이지 주소로 번역되었다. GUARD FTL은 이 과정을 축소하여, 애플리케이션이 보낸 파일 오프셋을 곧바로 물리 페이지 주소로 변역한다. 그러므로, 기존의 FTL중 보호 파티션 영역을 맵핑하는 영역은 필요없으므로 제거되고, 확보된 램 공간에 GUARD FTL을 포함한 파일 시스템을 올리기 때문에 추가적인 공간 오버헤드를 갖지 않는다.

우리는 시스템을 증명하기 위해, 리눅스 환경에서 SGX 라이브러리가 제공하는 IPFS를 수정하고, Jasmine OpenSSD [7] 플랫폼 내부에 디바이스 레벨 파일 시스템을 개발하여 SSD GUARD를 프로토타입으로 제작하였다. 실험 결과, SSD GUARD는 쓰기와 읽기 성능이 기존 IPFS에 비해 최대 33%, 2% 낮아졌다. 하지만, SSD GUARD가 기존에 IPFS가 제공하지 못한 보안성을 제공한다.

2 Intel SGX

Intel SGX로 빌드되는 유저 애플리케이션은 일부 코드 및 데이터 영역의 기밀성과 무결성을 보장받는다. 프로세서에 의해 하드웨어적으로 보호되는 해당 영역을 인클레이브라고 한다. SGX 기반의 애플리케이션은 보호 받지 않는 영역 (untrusted part)와 인클레이브로 구성된다. 먼저, 애플리케이션이 실행되면 untrusted part는 인클레이브를 동적으로 빌드하고, ECALL을 호출하여 인클레이브 내부 코드를 실행한다. 그런데 인클레이브는 직접 시스템콜을 호출하는 것이 제한된다. 따라서 하드웨어 자원을 이용하기 위해서 인클레이브는 우선 OCALL을 호출하여 untrusted part의 코드를 실행한다. untrusted part는 시스템콜을 호출 후 결과를 인클레이브에 리턴한다.

Intel IPFS는 인클레이브 내부에서 실행되는 SGX 라이브러리로, 파일의 기밀성과 무결성을 증명한다. IPFS는 파일을 쓸 때, 프로세서 또는 유저가 제공하는 파일 키를 기반으로 파일을 암호화하여 서명한다. 이후, OCALL을 호출하면, untrusted part는 시스템콜(write)을 호출하여 네이티브 파일 시스템을 실행한다. 네이티브 파일 시스템은 암호화된 파일을 스토리지에 저장한다. 이후, 파일을 읽을 때는, 먼저 OCALL로 시스템콜(read)을 호출하여 네이티브 파일시스템으로부터 파일을 읽어온다. 그리고 파일을 복호화하고 증명하는 과정을 거쳐 파일의 기밀성을 보장하고, 무결성 훼손 여부를 검증한다.

이와 같이 IPFS는 네이티브 파일 시스템에 의존하므로, 운영체제가 침입당하면 파일을 보호하기 어려워진다. 이러한 문제를 해결하기 위해, SSD GUARD는 스토리지 내부에 디바이스 레벨 파일 시스템을 구현한다. 이를 통해 IPFS가 시스템콜로 네이티브 파일 시스템에 요청하던 다수의 파일 연산을 스토리지에서 대신 수행한다. 이 과정에서 운영체제의 파일 시스템은 우회되며, 운영체제는 데이터를 스토리지로 전송하는 수동적인 역할만을 수행한다.

SSD GUARD는 스토리지 레벨에서 파일 시맨틱을 블록 주소로 번역하는 가벼운 파일 시스템을 제공한다. 하지만, SSD GUARD는 기존 네이티브 파일 시스템의 복잡한 기능을 모두 구현하지는 않는다. 우리의 목적은 유저가 IPFS가 제공하는 API로 파일을 방어하는 것이다. 이를 위해 SSD GUARD의 디자인 범위는 IPFS가 네이티브 파일 시스템에 요청하던 파일 연산을 대신 제공하는 것으로 제한된다. IPFS는 파일의 보호가 주목적이므로, 네이티브 파일 시스템에게 일부 단순한 파일 연산(열기, 닫기, 쓰기, 읽기 등)만 요구하기 때문이다. 예를 들면 IPFS는 unlink나 디렉토리 연산은 요청하지 않기 때문에, 디바이스 레벨 파일시스템은 해당 연산을 제공하지 않는다.

3 Threat Model

멀웨어는 사용자의 컴퓨터에 침입하여 유저 권한, 루트권한, 또는 가장 높은 권한 (Ring 0 level)까지 획득할 수 있다. 따라서 멀웨어는 유저 영역뿐 아니라, 운영체제 파일 시스템과 드라이버를 포함한 모든 영역을 침입할 수 있다. 하지만, 인클레이브는 프로세서에 의해 직접 보호되므로, 침입당하지 않는다고 가정한다. 또한, SSD 펌웨어 코드는 시큐어 부트와 디지털 서명으로 오직 신뢰할 수 있는 제조사만이 업데이트 할 수 있다고 가정한다. 따라서 SSD 펌웨어를 하나의 TEE로 본다. 우리는 공격 모델을 원격 공격자가 소프트웨어로 사용자의 컴퓨터를 침입하여 권한을 획득하고, 데이터를 변조하는 상황으로 제한한다. 즉, 우리가 제안하는 시스템은 공격자의 물리적인 공격을 방어할 수는 없다.

4 SSD GUARD

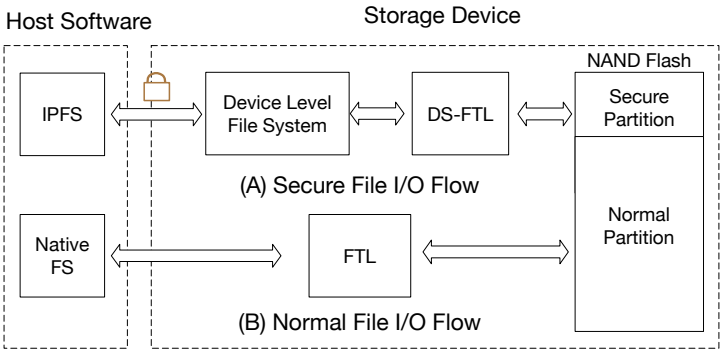

그림 1: 호스트의 파일 타입에 따라 선별적인 보안을 제공하는 스토리지 내부 설계

우리는 IPFS와 스토리지가 신뢰할 수 있는 TEE라는 점을 이용하여, 각 컴포넌트에 인증 모듈을 구현하여 안전한 커뮤니케이션을 보장한다. 안전한 커뮤네키이션은 다음과 같은 방식으로 설계된다. 먼저, IPFS와 스토리지는 인증 인클레이브의 도움을 받아 파일을 생성할 때 서로 파일 키를 공유하여 신뢰할 수 있는 커뮤니케이션 채널을 생성한다. 인증 인클레이브는 스토리지 제조사가 사용자에게 제공하는 SGX기반의 애플리케이션으로, IPFS와 스토리지가 서로 신뢰 채널을 구성하도록 돕는 보안 관리자이다. 이후 파일을 쓸 때, IPFS에서 전송되는 모든 데이터는 메세지 인증 코드 (MAC)을 이용하여 스토리지에서 인증되고, 성공 여부를 응답 값으로 IPFS에 리턴한다. IPFS에서 MAC을 이용하여 응답 값 인증에 성공하면, 파일 쓰기는 종료된다. 반면 스토리지로부터 실패 메세지를 받거나, 인증에 실패하면 사용자는 ERROR를 리턴받는다. 이번 논문은 SGX를 보조하기 위한 스토리지 설계에 초점을 맞추므로, 안전한 커뮤니케이션을 위한 인증 인클레이브와 파일 키 공유 과정의 상세한 구현은 논문 범위에서 벗어나므로 생략한다.

Figure 1은 SSD GUARD의 스토리지의 내부 설계를 보여준다. SSD GUARD는 단일 스토리지 안에서 파일의 중요도에 따라 선별적으로 파일을 보호하기 위한 메카니즘을 제공한다. 이를 위해 단일 스토리지 안에서 일반 파일과 보호 파일의 I/O를 동시에 지원하도록 설계되었다. 여기서 일반 파일은 네이티브 파일 시스템에서 블록단위의 I/O를 수행하는 기존의 파일을 의미한다.(그림 1(b)에 나타난다.) SSD GUARD는 일반 파일에 추가적인 보안을 적용하지 않는다. 반면 보호 파일은 SGX의 IPFS에서 전송한 파일로, 변조 공격으로부터 보호받는다.(그림 1(a)에 나타난다.) 이 설계 덕분에 사용자는 추가적인 스토리지를 구매할 필요가 없다. 제조사가 펌웨어 업데이트를 지원하기만 하면, 사용자는 기존에 사용하던 SSD를 이용하여 선별적으로 파일을 보호할 수 있기 때문이다.

펌웨어를 업데이트하기 전 기존의 일반적인 SSD는 일반 파일 I/O을 지원하기 위해 디램 공간에 FTL을 가진다. FTL은 네이티브 파일 시스템이 보낸 논리 블록 주소를 낸드 페이지의 물리 주소로 번역하는 테이블을 의미한다. 펌웨어를 업데이트하면 사용자는 낸드 플래시의 일부 영역을 보호 파티션 (Secure Partition)으로 할당할 수 있다. 보호 파티션은 오직 보호 파일만이 저장되므로, 네이티브 파일 시스템으로는 해당 영역에 접근할 수 없다. 따라서 디램 공간의 FTL중 보호 파티션의 물리적 주소를 가리키던 영역은 더이상 사용되지 않으므로 수거된다. 이 때 생겨난 여유 공간에는 보호 파일의 I/O를 수행하기 위한 디바이스 레벨 파일 시스템과 GUARD

FTL이 로드된다.

사용자가 일반 파일의 I/O를 요청하면, SSD GUARD는 기존의 일반적인 SSD와 같은 방식으로 I/O를 수행한다. 네이티브 파일 시스템에서 I/O를 위한 정보(논리 블록 주소, 크기, 명령어 타입, 데이터 버퍼)를 보내면, SSD GUARD는 FTL을 통해 논리 블록 주소를 낸드 페이지 주소로 변환하여 I/O를 수행한다. 이때 낸드 페이지 주소는 일반 파티션(Normal Partition)내부의 주소를 가리킨다.

추가적으로, SSD GUARD는 보호 파일의 연산을 위해 디바이스 레벨 파일 시스템과 GUARD FTL을 제공한다. 디바이스 레벨 파일 시스템은 기존 IPFS가 의존하던 네이티브 파일 시스템을 대체하는 파일 시스템으로, 파일 열기, 쓰기, 읽기, 닫기와 같이, IPFS가 필요로 하는 범위 내에서 기본적인 파일 오퍼레이션을 제공한다. 이를 위해 디바이스 레벨 파일 시스템은 키 단위의 디렉토리와 파일 테이블과 아이노드를 가진다. 키 단위의 디렉토리는 동일한 파일키를 가진 파일의 정보를 저장하는 디렉토리로, 파일의 이름과 아이노드 번호를 맵핑한다. 파일 테이블은 사용자가 보낸 파일 디스크립터로 아이노드로 검색한다. 아이노드는 파일의 정보를 저장하는 자료구조로, 파일 이름, 아이노드 번호, 크기 뿐 아니라 IPFS와 공유한 파일의 키, GUARD FTL의 위치를 가진다. GUARD FTL은 보호 파일의 I/O를 수행할 때, 파일 오프셋을 낸드 플래시의 물리적 주소로 맵핑시켜주는 FTL이다.

IPFS가 보호 파일(foo.txt)의 열기를 요청하면, 디바이스 레벨 파일 시스템은 먼저, IPFS가 전송한 파일 키에 대응하는 키 단위의 디렉토리를 찾아, 파일 이름(foo.txt)에 대응하는 아이노드번호를 찾는다. 그리고 낸드 플래시에 저장된 아이노드와 GUARD FTL을 메모리에 로드한다. 그리고 파일 테이블에 아이노드를 추가하고 파일 디스크립터를 받아서 사용자에게 리턴한다. IPFS는 이후 파일 디스크립터를 이용하여 파일 쓰기와 읽기를 요청한다. IPFS는 파일 쓰기와 읽기를 요청할 때 디바이스 레벨 파일 시스템에 인자로 파일 디스크립터, 파일 오프셋, 크기와 같은 정보를 보내준다. 디바이스 레벨 파일 시스템은 파일 디스크립터로 파일 테이블을 검색하여 아이노드를 찾는다. 그리고 아이노드가 가리키는 GUARD FTL에 접근한다. 마지막으로, GUARD FTL을 검색하여 파일 오프셋에 매칭되는 낸드 물리 주소를 검색하여, 물리 주소에 데이터를 쓴다. 파일 읽기 역시 이와 같은 플로우로 구현된다. 마지막으로 IPFS가 파일 닫기 연산을 요청하면, 스토리지는 파일 테이블로부터 파일 디스크립터에 매칭되는 아이노드와 GUARD FTL을 찾아 메모리로부터 수거한다. 그리고 파일 테이블로부터 파일 디스크립터를 수거한다.

5 실험

우리는 실험을 위해 Intel SGX가 제공되는 Intel i7-8700 CPU와 16GB RAM 컴퓨터를 이용하였다. 우리는 호스트 운영체제 (Linux 4.10.16)의 드라이버 (SATA2.0)코드를 일부 수정하여, SGX 애플리케이션이 SSD GUARD의 파일 시스템을 이용하기 위한 SATA 커맨드를 새로 정의하였다. 그리고 우리는 Jasmine OpenSSD 플랫폼(ARM7TDMI-S core, 96KB SRAM, 64MB DRAM, 64GB NAND Flash)을 활용하여, SSD GUARD를 위한 디바이스 레벨 파일 시스템과 인증 모듈을 구현하였다.

Table 1은 파일 I/O 패턴에 따른 IPFS와 SSD GUARD의 성능을 비교한 실험 결과를 보여준다. 여기서 IPFS는 기존 IPFS라이브러리와 커널의 ext4 파일 시스템을 이용하여, SGX애플리케이션으로 기본적인 openSSD에 I/O를 수행하는 워크로드를 의미한다. 반면, SSD GUARD는 데이터 변조 방어를 위해, 수정된 IPFS로 SSD GUARD의 디바이스 레벨 파일 시스템에 I/O를 요청하는 워크로드이다. 이 실험에서 우리는 64MB의 단일 파일을 미리 생성하고, 이에 대한 덮어 쓰기와 읽기를 각 순차 접근 패턴과 임의 접근 패턴으로 수행하였다. 실험 결과, 베이스라인과 SSD GUARD간의 읽기 성능은 거의 차이가 나지 않았다. 이는 SSD GUARD가 수정된 IPFS의 읽기 요청에 대해 SSD GUARD가 어떠한 모듈 인증도 수행하지 않기 때문이다. 반면에 쓰기 실험에서는 각 순차 쓰기, 임의 쓰기에서 약 16%, 33% 감소하는 모습을 보였다. 이는 수정된 IPFS의 쓰기 요청에 대해 SSD GUARD가 인증을 수행하면서 오버헤드가 발생하기 때문인 것으로 보인다.

표 1: 파일 I/O 패턴에 따른 Throughput 비교 (MB/S)

	Seq Read	Ran Read	Seq Write	Ran Write
IPFS	8.629	2.89	1.10	0.43
SSD GUARD	8.537	2.85	0.93	0.29

6 결론

본 논문에서는 커널 권한의 강력한 데이터 변조 시도를 방어하기 위한 스토리지 시스템, SSD GUARD를 제안한다. SSD GUARD는 커널 권한의 멀웨어로부터 취약한 커널 파일 시스템 대신, 안전한 스토리지 펌웨어에서 가벼운 디바이스 레벨 파일시스템을 구현한다. 이를 통해, 로컬 환경에서 TEE 기반의 애플리케이션이 변조 위협으로부터 자유로운 데이터 저장을 가능하게 한다. 우리는 우리의 시스템을 증명하기 위해, end-to-end 프로토타입을 제작하였다. 실험 결과, 읽기와 쓰기 성능이 베이스라인에 비해 최대 약 2%, 33%감소하였지만, 기존 IPFS가 제공하지 못한 보안성을 제공한다.

참고 문헌

[1] C. SAM, "2017-2019 ransomware statistics and facts." https://www.comparitech.com/antivirus/ransomware-statistics/, 2018.

[2] V. VITOR, "Wiper malware: Attacking from inside." https://talos-intelligence-site.s3.amazonaws.com/production/document_files/files/000/033/904/original/Talos_WiperWhitepaper.v3.pdf, 2018.

[3] D. Jain, "Shamoon 2: Back on the prowl." https://nsfocusglobal.com/shamoon-2-back-on-the-prowl/, 2017.

[4] "Petya/notpetya ransomware analysis." https://idafchev.github.io/writeup/2017/07/21/petya_ransomware_analysis.html, 2017.

[5] V. Costan and S. Devadas, "Intel SGX explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[6] Intel, "Intel protected file system library." https://software.intel.com/sites/default/files/managed/76/8f/OverviewOfIntelProtectedFileSystemLibrary.pdf.

[7] S.-P. Lim, "The jasmine openssd platform: Technical reference manual (v1.4, in english)." http://www.openssd-project.org/mediawiki/images/Jasmine_Tech_Ref_Manual_v.1.4e.pdf, 2016.