형태보존 암호를 이용한 PE File IAT 보호 기법

박도현, 장원영, 이선영*
순천향대학교
rharnr777@gmail.com, ozragwort@sch.ac.kr, *sunlee@sch.ac.kr

Method for Protection of PE File IAT Using Format-preserving Encryption

Dohyeon Park, Wonyoung Jang, Sun-young Lee*
Dept of information security Soonchunhyang Univ.

요 약

역공학에 취약한 구조는 프로그램 개발자의 지적재산권 침해와 정상 프로그램을 가장한 악성코드 제작에 악용될 수 있는 위험이 존재한다. 본 논문은 컴파일되어 실행되는 프로그램을 역공학으로부터 보호하기 위해 형태보존암호를 이용하여 IAT를 암호화하는 기법을 제안한다. 제안 기법은 실행코드의 구조를 변경하지 않고 호출 함수정보를 노출시키지 않기 때문에 이를 적용할 때 실행코드에 대한 역공학 방지가 가능함을 확인하였다.

Ⅰ. 서 론

컴파일되어 실행되는 프로그램은 모든 기계에 대해 독립적인 바이트코드로 구성된다. 이러한 형태의 코드는 역공학을 통해 원형에 가까운 코드로 복원될 수 있다[1]. 공격자는 이를 분석하고 프로그램의 로직을 변경하는 공격이 가능하다. 바이트코드로 존재하는 코드의 로직은 난독화가 적용되어 있더라도 역공학에 비교적 쉽게 노출될 수 있는 문제점이 있다[2]. 역공학에 취약한 구조는 개발자의 지적재산권 침해는 물론이고 정상 프로그램을 가장한 멀웨어 제작에도 악용될 수 있어 일반 사용자에도 많은 피해를 야기할 수 있다[3]. 본 논문에서는 실행 파일의 호출 API 목록을 보호하는 것을 목적으로 “형태보존암호를 이용한 PE File IAT 보호 기법”을 제안한다. 논문의 구성은 다음과 같다. 2장에서는 제안기법과 관련된 기술과 제안기법에서 사용하는 기술에 대한 설명으로 이어지는 장에서의 이해를 돕는다. 3장에서는 제안기법의 설계를 중심으로 제안기법을 적용함으로써 얻을 수 있는 효과를 설명한다. 4장에서는 제안기법의 적용시 기대효과와 함께 결론으로 끝을 맺는다.

Ⅱ. 사전연구

1.1. IAT(Import Address Table)

메모리는 명령어와 데이터를 저장하는 공간이다. 프로그램이 구동되려면 디스크에 이진 데이터로 저장된 파일을 주 메모리에 적재하여 프로세스가 되어야 한다[4]. IAT란 실행파일이 어떤 라이브러리에서 어떤 함수를 사용하는 지에 대해 기술되어 있는 테이블이다. 예를 들면 notepad.exe는 내부적으로 kernel32.dll, ntdll.dll, user32.dll 등에 있는 다수의 함수를 사용하고 있다[5]. 이러한 함수들의 이름은 이미 실행 파일 내에 저장되어 있고 이곳의 위치를 알려주는 것이 Import table이다. 애플리케이션이 필요로 하는 모든 DLL(Dynamic Link Library)을 찾기 위해 애플리케이션의 IAT을 파싱한 후 각 DLL의 IAT를 재귀적으로 파싱한다[5]. 일반적으로 하나의 실행파일은 여러 라이브러리를 쓰기 때문에 라이브러리의 수만큼 구조체의 배열이 존재하며 마지막 구조체 배열은 NULL 구조체로 끝난다[6]. 실행에 필요한 데이터들을 메모리에 올리는 작업을 통해 메모리상의 API 주소를 가져온다. 따라서 실행 중인 프로세스를 분석한다면 IAT에서 관찰되는 API만으로도 프로세스의 행위를 추측할 수 있다[7].

1.2. IAT에 대한 공격 및 보호기술

IAT 공간을 이용한 공격 방법으로 API 후킹이 있다. API 후킹은 프로그램의 실행 시에 동적으로 만들어진 IAT의 4바이트 주소 값만을 변경해서 프로그램의 로직을 변경하는 공격이다. 또한 디스크 상에 파일을 조작하지 않고 공격을 수행하기 때문에 흔적을 남기지 않는 공격을 수행할 수 있다. 후킹은 이미 작성되어 이는 코드의 특정 지점을 가로채서 동작 방식에 변화를 주는 기술이다. API 호출 전·후에 사용자의 훅 코드(Hook Code)를 실행하거나 API가 넘겨받은 파라미터 혹은 리턴 값을 보거나 조작할 수 있다. 그리고 API 호출 자체를 취소시키거나 사용자 코드로 실행 흐름을 변경할 수 있다[5]. 이러한 공격은 실행 파일의 역공학 시 IAT 주소가 보호되지 않기 때문에 발생할 수 있는 공격이며 현재 보호기법으로는 코드 난독화, 더미코드 삽입(polymorphic code) 생성을 통해 분석을 어렵게 하는 기법이 사용되고 있다[8]. 하지만 난독화 결과가 저장되는 메모리 주소를 연결해 복구하면 API 난독화를 무력화할 수 있기 때문에 완전한 IAT 보호기술로 활용할 수 없다.

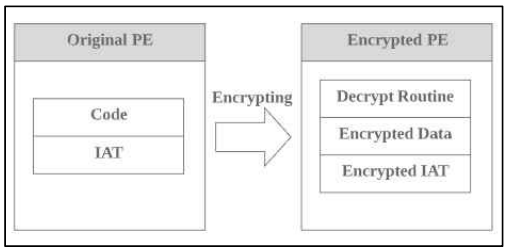
1.3. 형태보존암호

형태보존암호(Format-Preserving Encryption)는 평문의 형태와 길이를 그대로 유지하여 암호화하는 대칭키 암호방식이다. 일반적인 블록 암호화와는 달리 데이터와 연관된 부가정보인 트윅(Tweak)을 적용하여 암호화가 필요한 데이터의 길이를 유지할 수 있다. 문자 데이터에 대한 형태보존암호는 데이터베이스 암호화, 네트워크 전송 시 민감한 데이터 보호 등에 활용할 수 있다[9]. 예를 들어 16자리 신용카드 번호가 암호화되면 AES로 생성된 암호문은 이진 데이터 블록이 되지만 형태보존암호를 사

용한 암호문은 또 다른 16자리의 숫자가 된다.

Ⅲ. 형태보존암호를 이용한 IAT 암호화

실행 파일의 알고리즘을 역공학으로부터 보호하기 위해 형태보존암호를 이용하여 IAT를 암호화하는 방법을 제안한다. 윈도우즈 운영체제에서 사용하는 실행파일의 형식인 PE(Portable Executable)파일은 실행 파일에서 사용하는 외부 라이브러리 API들의 주소를 IAT에 저장한다. 운영체제는 DLL을 최초 한번만 메모리에 적재(loading)하고, 그 이후에는 프로세스에 매핑(mapping)한다. 따라서 디렉터리의 위치가 링크 시점에는 실행 시점마다 주소가 다르기 때문에 주소 정보가 하드코딩 될 수 없다. 이름 해석에는 호출자가 전체 경로를 명시하지 않고 함수 이름과 주소를 통해 메모리에 매핑한다. 프로세스 매핑과정에서 IAT를 참조하여 DLL이름과 주소를 얻는다[10].



[그림 1] 실행 파일 암호화 과정

암호화 과정은 [그림1]과 같다. 암호화를 위해 새로운 메모리를 쓰기권한으로 할당받고 원본 API의 주소와 이름을 IAT로부터 가져온다. 할당받은 메모리에 암호화를 수행한 결과를 새로운 IAT로 저장한다. 원본 API와 암호화가 적용된 주소를 연결하는 복호 루틴(Decrypt Routine)을 추가하여 작성하고 프로그램의 시작점(Entry Point)으로 수정한다.

Description	Value	Description	Value
Hint/Name RVA	0000 FindFirstFileW	Hint/Name RVA	0000 GetModuleHandleA
Hint/Name RVA	0000 GetFileAttributesW	End of Imports	kernel32.dll
Hint/Name RVA	0000 lstrcmpW	Hint/Name RVA	0000 RegQueryValueExW
Hint/Name RVA	0000 MulDiv	End of Imports	ADVAPI32.dll
Hint/Name RVA	0000 lstrcpynW	Hint/Name RVA	0000 CreateStatusWindowW
Hint/Name RVA	0000 LocalSize	End of Imports	COMCTL32.dll
Hint/Name RVA	0000 GetLastError	Hint/Name RVA	0000 PageSetupDlgW
Hint/Name RVA	0000 WriteFile	End of Imports	comdlg32.dll
Hint/Name RVA	0000 SetLastError	Hint/Name RVA	0000 EndPage
Hint/Name RVA	0000 WideCharToMultiByte	End of Imports	GDI32.dll
Hint/Name RVA	0000 LocalReAlloc	Hint/Name RVA	0000 _XcptFilter
Hint/Name RVA	0000 FormatMessageW	End of Imports	msvcrt.dll
Hint/Name RVA	0000 GetUserDefaultUILanguage	Hint/Name RVA	0000 DragFinish
Hint/Name RVA	0000 SetEndOfFile	End of Imports	SHELL32.dll
Hint/Name RVA	0000 DeleteFileW	Hint/Name RVA	0000 GetClientRect
Hint/Name RVA	0000 GetACP	End of Imports	USER32.dll
Hint/Name RVA	0000 UnmapViewOfFile	Hint/Name RVA	0000 GetPrinterDriverW
Hint/Name RVA	0000 MultiByteToWideChar	End of Imports	WINSPOOL.DRV
Hint/Name RVA	0000 MapViewOfFile		
Hint/Name RVA	0000 UnhandledExceptionFilter		
End of Imports	KERNEL32.DLL		

[그림 2] 형태보존 암호화 적용 전·후 Import table

[그림 2]는 암호화 적용 전, 후의 Import table을 'PE view' 상용 도구를 이용하여 정적 분석한 것이다. 적용한 암호화로 인해 정적 분석으로는 내부에 숨겨진 원본파일의 IAT에 대한 정보를 알 수 없다. 실행파일 실행 시 암호화로 인해 로더는 암호화된 API의 주소를 IAT에 작성한다. 형태보존암호를 사용하면 호출할 API 주소와 이름만을 복호하여 사용한다. 암호화된 파일이 실행되었을 때 메모리에 매핑된 후 복호 루틴에 의해 실행 압축이 해제되고 원래의 프로그램이 실행된다. 실행파일이 실행된 후 복호 루틴이 완료되면 원본 파일의 흐름으로 넘어가는 시점인 OEP(Original Entry Point)로 진행한다. 하지만 OEP를 찾아내 덤프를 했다고 해도 IAT를 정상적인 상태로 복구시키지 않으면 파일이 실행되는데 필요한 함수를 Import 할 수 없기 때문에 정상적인 실행이 불가능하다. 이러한 암호화 방법은 바이러스와 같은 실행코드 끼워 넣기를 방지하는 효과가 있으며 실제 실행파일의 구조를 변경하지 않고 실행코드의 호출 함수정보를 노출시키지 않기 때문에 실행코드에 대한 역공학이 어렵다는 장점을 가진다. 실행 파일이 필요한 함수를 호출할 때마다 DLL의 전체 API를 메모리에 매핑하는 기존 실행 방법에 비해 다른 API의 주소와 이름을 보호할 수 있다.

Ⅳ. 결론

본 논문에서는 API 난독화와 형태보존암호를 이용한 IAT 암호화를 통해 PE파일을 보호하는 방법을 제안하였다. 실행파일의 핵심 아이디어를 역공학으로부터 보호하고 프로그램이 메모리에 로드될 때 실행 가능한 형태로 복구한다. 암호화된 파일의 정적분석에서 높은 저항성을 갖는 것을 확인하였다. 원본파일의 정보를 숨기기 위해 실행파일 보호 기법인 암호화와 난독화를 적용하였고 OEP이후에도 분석방지 기술을 적용하여 역공학을 어렵게 하였다. 형태보존암호를 적용해 IAT 사용 시 해당하는 주소와 이름 영역만을 복호화하여 메모리에 매핑하여 사용할 수 있음을 확인하였다. 현재 더미코드 삽입 등 다양한 역공학 방지 기술이 개발되어 있으며 제안한 기법과 기존의 기법을 함께 사용한다면 더 강력한 난독화 기법으로 사용할 수 있을 것으로 기대된다.

ACKNOWLEDGMENT

이 성과는 정부(교육과학기술부)의 재원으로 한국연구재단의 자원을 받아 수행된 연구임(No. NRF2018R1D1A1B07047656).

참 고 문 헌

[1] 오진태, 장종수, 류재철, "제로데이 어택 방지를 위한 실시간 대응기술." 정보과학회논문지 v.37, no.4, pp. 255-262, 2010.

[2] Y. Liao, J. Li, B. Li, G. Zhu, Y. Yin, and R. Cai, "Automated Detection and Classification for Packed Android Applications", In 2016 IEEE International Conference on Mobile Services, pp. 200-203, 2016.

[3] J. H. Jung, J. Y. Kim, H. C. Lee, and J. H. Yi, "Repackaging Attack on Android Banking Applications and Its Countermeasures", Wireless Personal Communications, Vol. 73, No. 4, pp. 1421-1437, 2013.

[4] 황상엽, "파일 암호화 기반 랜섬웨어 탐지에 대한 연구", M.A. 숭실대학교, 2017.

[5] 이승원, "리버싱 핵심원리", 인사이트, 2012.

[6] 강병탁, "리버스 엔지니어링 바이블", 위키북스, 2012.

[7] Pavel Yosifovich, Alex Inoescu, Mark Russinovich, David Solomon. "Windows Internals, Part 1: System architecture, processes, threads, memory management, and more", 에이콘, 2018.

[8] 이재휘, 이병희, 조상현, "최신 버전의 Themida가 보이는 정규화가 어려운 API 난독화 분석방안 연구", KIISC, 2019.

[9] 박도현, 김민태, 임종훈, 장래승, 장원영, 이선영, "형태보존암호를 이용한 IPv6 라우팅 헤더 보호 기법 제안", 한국통신학회 동계종합학술발표회, Vol. 71 No. 1, pp. 757-758, 2020.

[10] Lee Jae-hwi, Han Jaehyeok, Lee Min-wook, Choi Jae-mun, Lee Sangjin, "A Study on API Wrapping in Themida and Unpacking Technique", Journal of the Korea Institute of Information Security and Cryptology, vol. 27, pp. 67-77, 2017.