논문 2020-2-2 http://dx.doi.org/10.29056/jsav.2020.12.02

MS Windows에서 인젝션 공격 및 방어 기법 연구

성호준**, 조창연***, 이호웅****, 조성제*†

A Study on Injection Attacks and Defenses on Microsoft Windows

HoJun Seong**, ChangYeon Cho***, HoWoong Lee****, Seong-Je Cho*†

요 약

기업이나 기관의 데스크톱 및 엔터프라이즈 서버용 운영체제로 마이크로소프트사의 Windows가 많이 활용되고 있고 사이버 공격의 주요 대상이 되고 있다. 마이크로소프트사는 다양한 보호 기술을 제공하고 주기적인 보안 패치를 통해 노력하고 있지만, 여전히 DLL 인젝션(injection)이나 프로세스 인젝션 등의 공격 위협이 존재하고 있다. 본 논문에서는 Windows 시스템에서 12가지 인젝션 공격 기법에 대해 분석하고, 4개의 응용 프로그램들을 대상으로 인젝션 공격 실험을 수행한다. 실험 결과를 통해 인젝션 공격의 위험성을 파악하고, 마이크로소프트에서에서 제공하는 인젝션 공격에 대한 완화 기술의 유효성을 검증한다. 실험 결과, 현재 응용 프로그램들이 여러 인젝션 공격에 취약함을 알 수 있었다. 최종적으로, 이러한 인젝션 공격에 대한 완화 기법을 제시하고 효용성을 분석하였다.

Abstract

Microsoft's Windows system is widely used as an operating system for the desktops and enterprise servers of companies or organizations, and is a major target of cyber attacks. Microsoft provides various protection technologies and strives for defending the attacks through periodic security patches, however the threats such as DLL injection and process injection still exist. In this paper, we analyze 12 types of injection techniques in Microsoft Windows, and perform injection attack experiments on four application programs. Through the results of the experiments, we identify the risk of injection techniques, and verify the effectiveness of the mitigation technology for defending injection attacks provided by Microsoft. As a result of the experiments, we have found that the current applications are vulnerable to several injection techniques. Finally, we have presented the mitigation techniques for these injection attacks and analyzed their effectiveness.

한글키워드 : Windows 운영체제, 프로세스 인젝션, DLL 인젝션, 완화 정책, 방어 기술

keywords : Windows OS, Process Injection, DLL Injection, Mitigation Policy, Defense Technology

* 단국대학교 컴퓨터학과
** 단국대학교 응용컴퓨터공학과
*** 단국대학교 소프트웨어학과
**** 호서대학교 컴퓨터정보공학부
† 교신저자: 조성제(email: sjcho@dku.edu)
접수일자: 2020.11.15. 심사완료: 2020.11.30.
게재확정: 2020.12.21.

1. 서 론

마이크로소프트사의 Windows는 그래픽 사용자 인터페이스(GUI)와 멀티태스킹(multitasking)

을 지원하는 운영체제(OS)로 전 세계 90%의 개인용 컴퓨터 및 서버용 OS로 많이 활용되고 있다[1]. MS Windows OS가 널리 사용됨에 따라 이를 대상으로 하는 보안 공격도 많이 발생하고 있다. 예로, 코드 인젝션(code injection), DLL 인젝션(DLL injection)[2][3], 프로세스 인젝션(process injection)[4][5]과 공격 유형들은 많은 기업과 기관, 일반 사용자들에게 많은 피해를 주고 있다. 이러한 공격 유형은 Windows API(Application Programming Interface)를 사용해 공격 흔적을 남기지 않고 피해자 컴퓨터에 접근하여 중요한 데이터를 획득한 후 유출하기도 한다. 또한, 인젝션 공격은 랜섬웨어(ransomware)와 함께 악용되는 사례도 나타나고 있다. 대표적인 예로 2017년 기업의 서버와 클라이언트를 대상으로 한 Sorebrect 랜섬웨어와 2019년 물류센터나 공공기관의 기업 네트워크를 대상으로 한 NetWalker 랜섬웨어가 있다[6][7]. DLL 인젝션 기법을 활용하는 NetWalker는 Mailto 랜섬웨어로도 알려져 있으며, 2020년 1분기 최악의 신규 랜섬웨어로 분류되었다[8][9]. NetWalker는 정상 프로그램으로 위장하기 위해 반사 DLL 인젝션(reflective DLL injection) 기법[10]이 적용된 파일리스 악성코드[11][12]다.

레드 팀 보안 업체로 잘 알려진 Red Canary사는 MITRE ATT&CK (Adversarial Tactics, Techniques and Common Knowledge)의 다양한 공격 기법을 기반으로 2019년 한 해 동안 적발된 보안 위협 상위 20가지를 공개하였다[13][14]. Red Canary의 2019년 위협 탐지 보고서에 따르면, 여러 위협 중 프로세스 인젝션 기법이 17%로 가장 위협적임을 보여준다[13].

본 논문에서는 Windows OS에서 프로세스 인젝션 등의 위협이 얼마나 심각한지 평가하고, 이들 위협에 대한 방어 기법에 대해 연구한다. 프로세스 인젝션 기법의 영향을 평가하기 위해, 먼저 4개 응용 프로그램으로 계산기(calculator), 노트패드(notepad), CCS(code composer studio), 파일 탐색기(explorer)들 선정한다. Windows 10의 2가지 빌드 환경에서 선정한 프로그램들을 대상으로, SafeBreach사에서 C/C++ 라이브러리 형태로 개발한 Pinjectra[15]를 사용해 인젝션 공격을 진행하여 평가한다. 안전한 실험을 위해 모든 과정은 가상머신에서 진행하며, Process Explorer[16]를 사용하여 응용 프로그램에 대한 정보를 분석한다. 또한, 인젝션 공격을 방어하는 기존 방어 기술들을 살펴보고, 마이크로소프트사에서 제공된 인젝션 방어 기술의 유효성을 검증한다.

본 논문의 구성은 다음과 같다. 2장에서 배경지식 및 관련연구에 대해 설명하고, 3장에서 인젝션 공격의 적용 및 평가 방안에 대해 기술한다. 4장에서 프로세스 인젝션 공격기법들을 사용하여 실험하고, 5장에서는 인젝션 공격에 대한 대응 기술들의 효용성을 분석하고, 6장에서 결론을 맺는다.

2. 배경지식 및 관련연구

2.1 Windows에서 인젝션 공격

인젝션 공격은 4가지 단계로 진행된다. 첫 번째로, 대상 프로세스에 접근한다. 두 번째로, 대상 프로세스에 임의의 메모리 공간을 할당한다. 세 번째로, 할당한 메모리 공간에 쓰기 기법을 사용해 대상 프로세스에 코드를 쓰는 작업을 진행하고, 마지막으로 대상 프로세스 또는 스레드를 실행하면 공격이 수행된다.

API 호출을 기준으로 프로세스 인젝션 공격을 4가지 유형으로 그룹화 한 것이 [17]에 나타나 있다. [17]의 인젝션 기법들은 대상 프로세스에서

임의 코드를 실행하는 목적을 가지며, 섹션(Section), 프로세스, 코드, DLL의 형태로 코드가 대상 프로세스에 주입된다.

섹션 인젝션은, 섹션 형태로 코드가 주입되는 것을 말한다. 섹션은 프로세스간 공유될 수 있고 특정 API를 사용하여 생성할 수 있는 메모리 블록을 말한다. 또한 섹션은 프로세스가 파일을 메모리 주소 공간에 매핑할 수 있는 기능을 제공한다. 이에 따라 섹션을 매핑 해제하고 악성코드를 주입하고 다시 매핑하는 행위를 통해 섹션 인젝션 공격을 수행할 수 있다. 섹션 인젝션 외에 프로세스 인젝션, 코드 인젝션, DLL 인젝션은 다음과 같다.

2.1.1 프로세스 인젝션

MITRE ATT&CK에 따르면, 프로세스 인젝션은 실행 중인 프로세스의 주소 공간에서 임의의 코드를 실행하는 것을 말한다[14]. 희생 프로세스의 문맥(context)에서 주입된 코드를 실행하면 해당 프로세스의 메모리, 리소스 및 상승된 권한에 접근할 수 있다. 메모리나 상승된 권한에 접근함으로써 악성 행위를 수행할 수 있다. 프로세스 인젝션을 통한 실행은 합법적인 프로세스에 의해 실행이 마스킹(masking)되므로 보안 제품의 탐지를 피할 수 있다는 특징이 있다.

2.1.2 코드 인젝션

Kaspersky Lab의 Encyclopedia에 따르면 코드 인젝션은 임의 코드(악성 코드)를 실행하기 위해 취약한 프로그램을 조작하는 행위를 말한다. 코드 인젝션은 안전하지 않은 사용자가 제공하는 데이터를 실행 중인 대상 프로세스의 코드의 일부가 되도록 허용하는 취약점이 존재할 때 발생하며, 이로 인해 악의적인 결과에 사용될 수 있다[18].

2.1.3 DLL 인젝션

Kaspersky Lab의 Encyclopedia에 따르면 DLL 인젝션은 타사 DLL을 로드하여 실행 중인 프로세스에 코드를 주입하는 방법이다. 이를 위해 공격자는 실행중인 프로세스들 중 하나에 새로운 스레드를 생성하고 해당 프로세스에 자신의 DLL을 추가한다. 이 방법은 cheat(컴퓨터 게임에서 해당 게임을 하지 않고 그 다음 단계로 바로 넘어갈 수 있는 프로그램) 작성과 같이 온라인 게임에서 널리 사용되고, 시스템 기능을 가로채는 등, 기타 악의적인 행위를 수행하는 데에도 사용될 수 있다[19].

2.2 Linux에서 인젝션 공격

본 논문에서는 Windows에서의 인젝션 공격을 주로 다루고 있기 때문에 간단하게 Linux에서의 인젝션 공격에 대해 살펴본다.

먼저, Ptrace 시스템 호출 인젝션 기법이다. Ptrace는 시스템 호출이며, 이를 통해 컨트롤러가 대상 프로세스의 상태를 조사하고 조작할 수 있다. 실행중인 대상 프로세스에 임의의 코드를 작성한 다음 PTRACE_SETREGS로 호출하여 다음으로 실행할 명령어를 포함하는 레지스터로 설정해줌으로써 수행된다. 권한이 높은 프로세스를 대상으로는 수행할 수 없고, PTRACE_POKETEXT나 PTRACE_POKETDATA를 사용하여 수행할 수도 있다[20]. 이외에도 Ptrace를 사용하여 .so 파일을 인젝션하는 기법도 있다.

다음으로, Proc 메모리 인젝션은 일반적으로 /proc 파일 시스템을 통해 프로세스의 메모리를 열거한 다음, 사용가능한 가젯(gadget)과 명령으로 ROP(Return-Oriented Programming) 페이로드를 구성하고 메모리 매핑을 사용하여 대상 프로세스의 스택을 덮어쓰는 방식으로 수행된다[21].

2.3 인젝션 공격의 완화 기술

Windows 환경에서 인젝션 공격에 대응하기 위해 마이크로소프트사에서는 여러 완화 기술들을 제공하고 있다. 마이크로소프트가 제공하는 보안 기능들 중에서, CFG(Control Flow Guard), CIG(Code Integrity Guard), ACG(Arbitrary Code Guard)에 대해 살펴본다.

CFG는 메모리 손상(memory corruption) 취약점을 방어하기 위한 보안 기능으로, 제어 흐름이 예기치 않은 위치로 재지정(redirection)되는 것을 방지하는 기술이다. 대상 주소가 유효한지 확인하기 위해 모든 간접 호출 전에 검사하는 코드를 삽입한다. 애플리케이션 코드를 실행할 수 있는 위치에 엄격한 제한을 두면 버퍼 오버플로우(메모리를 다루는 데에 오류가 발생하여 잘못된 동작을 하는 프로그램 취약점)와 같은 취약점을 통해 임의 코드를 실행하는 것이 훨씬 더 어려워진다. 이 기능은 Windows(8, 10)에 대한 마이크로소프트사의 CFI(Code Flow Integrity) 개념을 구현한 것이다[22][23][24].

CIG는 코드 서명 제약(Code Signature Constraint)으로, 적절하게 서명된 코드만이 메모리로 적재되게 하여 공격자 DLL이 적재되지 않게 하며, Device Guard 정책에 의해 구성될 수 있다. Device Guard는 Windows 하이퍼 바이저(사용자 컴퓨터에서 다수의 운영체제를 동시에 실행하기 위한 논리적 플랫폼)를 사용하여 장치에서 보안 서비스를 지원함으로써 가상화 기반 보안을 가능하게하는 Windows 10 보안 기능으로 보안 부팅, UEFI(Unified Extensible Firmware Interface : 운영체제와 플랫폼 펌웨어 사이의 소프트웨어 인터페이스를 정의하는 규격) 잠금 및 가상화와 같은 보안 기능을 활성화하는 정책들로 구성되어 있다. 현재 CIG를 지원하는 애플리케이션은 Microsoft/Microsoft Store/ WHQL에 의해 서명된 모듈(DLL, 바이너리)만 프로세스 메모리로 적재한다. 이 기능의 주요 이점은 사용자의 컴퓨터가 감염된 경우에도 악성코드가 CIG로 보호되는 앱에 악성코드를 주입할 수 없다는 것이다[25].

ACG는 서명된 코드 페이지들이 변조되지 않게, 또한 서명되지 않은 페이지들이 생성될 수 없게 보장한다. 프로세스의 동적 코드 정책으로 설정 시 Windows 커널은 프로세스가 코드 생성/수정/실행하는 것을 방지한다. 코드에 대해서는 쓰기 연산을 할 수 없어 변조가 불가능하고, 데이터에 대해서는 실행허가가 없고 쓰기가 가능하다. 이는 더 이상 Windows API인 VirtualProtect와 VirtualAlloc을 사용하여 새로운 PAGE_ EXECUTE_READWRI TE 권한을 가진 페이지를 생성할 수 없음을 의미한다[25].

5 장에서는, 이 3가지 기능을 적용하여 인젝션 공격을 어느 정도 완화할 수 있는지 분석한다.

2.4 기존 방어 기법 조사

다음은 인젝션 공격에 대해 잘 알려진 방어 기술로써 해시를 이용한 메모리 영역 비교, API 후킹, 포렌식 분석을 통한 데이터 기반 블랙리스트 구축에 대해 설명한다.

2.3.1 해시를 이용한 메모리 영역 비교

임수민 등은 프로세스 Hollowing 기법에 대한 방어 기술[26]을 제시하였다. 인젝션 공격이 의심되는 프로세스를 동일한 실행조건으로 복제하고, 각 프로세스의 가상 메모리 영역에 속해 있는 데이터 집합을 추출한다. 그리고 퍼지 해시를 통해 유사도를 산출한다. 결과적으로 유사도가 특정 범위를 초과하는 경우를 감염된 척도로 판단하여 프로세스 Hollowing을 실시간으로 탐지하는 방법을 제시하였다.

2.3.2 API 후킹

API 후킹(API hooking)을 사용하여 인젝션 공격을 탐지하는 연구들이 수행되었다. [17]에 따르면 API 후킹을 통해 함수 호출의 제어권 획득이 가능하므로 API 모니터링으로 함수의 모든 인자에 대해 관찰이 가능하다. 단, Windows API의 특성상, 하나의 API 함수에 대해 다양한 래퍼 함수가 존재하기 때문에 function call에 계층적 구조를 분석하고, 효과적인 후킹을 위해서 가장 낮은 계층에 위치한 래퍼 API 함수에 Hook을 설치하는 것이 좋다. 이러한 로직을 바탕으로 UnRunPE[28]를 개발하였고, 이를 API 모니터링 이론을 실제 적용하기 위해 만들어진 PoC(Proof of Concept)라 설명하고 있다. 이를 더 발전시켜, 다양한 코드 인젝션 탐지가 가능한 Dreadnought[29]가 제안되었다. Dreadnought는 휴리스틱을 추가하여 코드 인젝션을 탐지한다.

Captain[27]은 API 후킹을 통해 악성 이벤트를 발견하고 위협 분석 프로세스를 개선하는 엔드포인트 모니터링 도구이다. 새로운 프로세스 생성 시 일부 Windows API 함수를 후킹하는 DLL을 삽입한다. 기존에 수집한 시그니처를 기반으로 일치되는 패턴을 찾는 방식으로 탐지를 수행한다.

2.3.3 포렌식 분석을 통한 데이터 기반 블랙리스트 구축

포렌식 도구를 사용하여 감염된 프로세스에 대해 분석을 진행하고 분석과정에서 획득한 데이터를 기반으로 블랙리스트를 구축하여 엔트리 포인트 또는 엔드 포인트에서 인젝션 공격을 탐지하는 방법들이 제시되었다. 이들 연구들은 포렌식 도구인 Rekall[30]이나 Volatility[31]를 활용하여 감염을 탐지하는 과정과 방법을 제시하였다.

[32]은 Rekall을 사용하여 VAD(Virtual Address Descriptor) 정보와 결합된 프로세스 스택 데이터를 사용하여 프로세스 실행에서 이상을 탐지한다. 또한, VAD의 메모리 보호 태그가 정상으로 보이도록 수정된 경우에도 스택 분석을 사용하여 코드 인젝션을 탐지할 수 있다. [33]는 Volatility를 사용하여 [32]과 동일하게 VAD 정보에 초점을 두고 인젝션 공격을 탐지한다. [34]은 공격 은닉 기법을 극복하면서, 페이지가 실제 실행 가능한 상태인지를 확인하고, 현재 엑세스할 수 없는 메모리와 아직 할당되지 않은 메모리를 구별하기 위해 VAD 대신 페이징 구조를 통해 나열되는 PTE(Page Table Entry)를 검사한다. 이는 VAD를 사용하는 것보다 빠르고 안정적이다.

3. 인젝션 공격의 적용 및 평가 방안

3.1 인젝션 공격 적용

본 절에서는 인젝션 공격들이 최신 운영체제인 Windows 10에서도 여전히 위협적인지를 평가하기 위해, 실제로 많이 사용되는 애플리케이션들을 대상으로 인젝션 공격을 수행하고 공격의 적용 여부를 분석한다. 인젝션 공격은, 쓰기 기법을 사용하여 대상 프로세스에 악의적인 코드를 삽입한 후 실행하는 것으로, 본래는 희생자가 공격당한다는 사실을 모르게 해당 프로세스를 감염시키는 등 은밀하게 공격이 진행된다. 본 논문에서는 인젝션 공격 적용 시에 공격 성공 여부를 확인할 수 있게, 공격 성공 시 메시지 박스 혹은 간단한 메시지 등을 출력하는 방식을 적용한다.

3.2 Pinjectra의 인젝션 공격 유형

Pinjectra[15]는 64-비트 Windows 10 운영체제에 초점을 맞춰 “mix and match” 방식으로 프

표 1. Pinjectra의 12가지 프로세스 인젝션 기법과 각 기법에 사용된 API 목록
Table 1. Twelve process injection techniques of Pinjectra and List of APIs used in each technique

Demo ID	Name	APIs
1	Windowshook Demo	SetWindowsHookEx, VirtualAllocEx, GetProcAddress
2	CreateRemoteThread + DLL Load	OpenProcess, WriteProcessMemory, CreateRemoteThread/OpenThread, QueueUserAPC/ ntdll!NtQueueApcThread
3	CreateRemoteThread + Code Injection Demo	OpenProcess, CreateFileMapping, MapViewOfFile, ntdll!NtMapViewOfSection, CreateRemoteThread
4	Thread Execution Hijacking and Variant #1	OpenProcess, WriteProcessMemory, OpenThread, SuspendThread, ResumeThread, SetThreadContext
5	QueueUserAPC + AtomBombing	OpenThread, GlobalAtomAdd, QueueUserAPC/ ntdll!NtQueueApcThread
6	Ctrl Inject	OpenProcess, WriteProcessMemory, ntdll!RtlEncodeRemotePointer, SendInput, PostMessage
7	ALPC	OpenProcess, WriteProcessMemory, VirtualQueryEx, NtDuplicateObject, NtConnectPort, ReadProcessMemory
8	Propagate	OpenProcess, WriteProcessMemory, FindWindows/OpenWindows, GetProp, SetProp, ReadProcessMemory
9	StackBomber	OpenThread, GetThreadContext, SetThreadContext, ntdll!NtQueueApcThread
10	SetWindowLongPtrA	OpenProcess, WriteProcessMemory, FindWindows/OpenWindows, SetWindowLong/ SetWindowLongPtr
11	SIR + GhostWriting	OpenThread, GetThreadContext, SetThreadContext, SuspendThread, ResumeThread
12	Unmap Map	OpenProcess, CreateFileMapping, MapViewOfFile, ntdll!NtMapViewOfSection, ntdll!NtUnmapViewOfSection, ntdll!NtSuspendProcess, ntdll!ResumeProcess, FlushInstructionCache, ReadProcessMemory

로세스 인젝션 기법들을 구현한 C/C++ 라이브러리다. Pinjectra는 2019년 8월 기준으로 “Stack Bomber”의 유일한 구현이기도 하다. Pinjectra에는 다양한 쓰기 기법과 실행 기법들이 존재한다. 이 기법들을 “mix- and-match”하게 되면 다양한 공격 기법을 구현할 수 있다. 표 1과 같이 총 12개의 인젝션 기법들로 구성되어 있는데[15], 과거에 사용되었던 기법부터 최근에 등장한 쓰기 기법과 실행 기법을 포함하고 있다. 어떤 API를 사용했는지에 따라 인젝션 공격들이 구분된다.

표 1에서, Pinjectra의 실제 Demo 프로그램의 CLI(Command Line Interface) 창에 출력되는 각 12가지 기법들의 이름을 ‘Name’ 열에 표시하였다. Demo ID 1번 “WindowsHook Demo”의 경우, 고전적인 실행 기법으로 쓰기 기법 없이 SetWindowsHookEx()만 사용하여 대상 프로세스에 DLL을 주입한다. Demo ID 2번 “CreateRemoteThread()+DLL Load”는 WriteProcessMemory()를 쓰기 기법으로 사용하여 대상 프로세스에 DLL을 쓰고, CreateRemoteThread()를 사용하여 스레드를 실행시켜 줌으로써 DLL의 코드를 실행한다. 나머지 10가지의 기법들도 앞서 설명한 2가지 기법들과 마찬가지로 2.1절에서 언급한 4단계로 진행된다.

Pinjectra는 인젝션 성공하면, 메시지나 메시지 박스와 같은 심볼(symbol)을 출력하여 주입 공격의 성공 여부를 나타낸다. 이를 통해 직관적으로 주입 공격의 성공과 실패 여부를 판단할 수 있다.

인젝션 공격 평가는 다음과 같이 4단계로 진행한다. 먼저, 대상 프로세스를 실행시킨다. 그리고 툴을 사용하여 대상 프로세스의 PID(Process ID)와 TID(Thread ID)를 획득한다. 획득한 PID와 TID 정보를 Pinjectra 실행 시 옵션 사항으로 함께 입력한다. Pinjectra가 실행되면 메시지나 메시지 박스를 통해 공격의 성공 여부를 알려준다.

4. 실 험

본 절에서는 Windows 10의 2가지 OS 빌드 환경에서 희생자 프로세스(victim process)들에 대해 Pinjectra 기반 프로세스 인젝션 공격을 진행한다. 희생자 프로세스로는 계산기(calculator), 메모장(notepad), CCS(code composer studio), 파일탐색기(explorer) 선정하였다. 인젝션 공격으로 발생할 수 있는 피해를 예방하고 안전성을 고려하여 고립된 가상머신 환경에서 공격을 진행한다. 가상머신으로는 VMware Workstation Pro 15를 사용하는데, 두 가지 유형의 실험 환경을 구축하기 위해 2개 가상머신에 각각 다른 빌드 버전의 Windows 10 이미지 파일을 사용한다. 표 2에서는 실험에서 사용하는 가상머신과 Windows 빌드 버전들의 하드웨어 사양을 정리하였다.

표 2. 실험 환경
Table 2. Experiment environment

	실험 환경 1	실험 환경 2
OS	Windows 10 Pro x64	
Version	1903	2004
Build	18362.959	19041.388
Processor	Intel Core i5-4670	Intel Core i5 -4690
RAM	4 GB	4 GB

표 2의 “실험 환경 1”에서는 2019년 5월경에 릴리즈된 1903 버전을 사용하고, “실험 환경 2”에서는 2020년 5월경에 릴리즈된 2004 버전을 사용한다.

4.1 인젝션 공격 및 모니터링 도구 설정

Pinjectra[15]를 실행하기 위해서 Visual Studio 2019를 통해 64 비트로 빌드하여 실행 파

일을 생성한다. CMD를 통해 Pinjectra를 실행하면 그림 1와 같이 사용법과 12가지 프로세스 인젝션 기법들을 출력하는 것을 확인할 수 있다. Pinjectra 사용 시 PID(Process ID)와 TID (Thread ID)가 필요하다. 이를 위해 마이크로소프트사에서 제공하는 프리웨어 작업관리자 및 시스템 모니터인 Process Explorer를 사용한다. Process Explorer는 PID와 TID 정보뿐만 아니라 프로세스에 로드되는 DLL 정보, CPU 사용량, 프로그램 이미지 종류 등과 같은 다양한 정보를 제공해준다.

그림 1. Pinjectra 실행 화면
Fig 1. Screen of displaying Pinjectra execution

Process Explorer를 통해 PID와 TID를 획득하는 방법은 다음과 같다. 먼저 PID/TID를 획득하고자 하는 프로그램을 실행시키고, Process Explorer 상에서 해당 프로세스를 더블 클릭을 하게 되면 그림 2와 같이 해당 프로세스에 대한 상세정보를 보여주는 속성창이 나타나게 된다. 상태 표시줄을 보면 프로세스 명과 PID 정보를 확인할 수 있으며, 여러 가지 속성 탭 중 Threads를 보면 몇 가지의 스레드가 실행되는 것을 확인할 수 있다. 여기서 우리는 확인하고자 하는 메인 스레드의 TID를 확인한다. 이렇게 Process Explorer를 통해 획득한 PID와 TID를 Pinjectra의 명령 옵션으로 입력하면 된다.

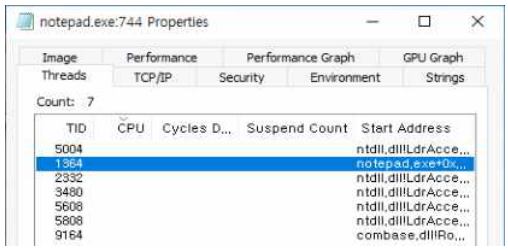

그림 2. Process Explorer에서 PID, TID 확인
Fig 2. Identifying PID and TID via Process Explorer

4.2 희생자 프로세스

실제 인젝션 공격에 사용될 희생자 프로그램(victim program)들은 다음과 같다.

- 파일 탐색기(Explorer.exe)
- 메모장(Notepad.exe)
- 계산기(Calculator.exe)
- CCS(Code Composer Studio) (CCStudio.exe)

파일 탐색기(Explorer.exe)는 [15]에서 언급한 프로그램으로, 파일 탐색기에 대한 인젝션 공격이 실제로 적용할 수 있는지 확인하기 위해 선정하였다. 메모장(Notepad.exe)은 다양한 사이버 공격 실험에서 사용되는 희생자 프로세스이다. 메모장은 단일 스레드를 사용하여 다른 멀티 스레드 프로세스들보다 가볍고 복잡하지 않아 인젝션 공격 실험에 적용할 수 있다.

그림 3. Windows 7에서 Explorer.exe 하위 프로세스에 위치한 Calc.exe
Fig 3. Calc.exe located in Explorer's subprocess in Windows 7

계산기(Calculator.exe)의 경우, Windows 7 환경에서는 Explorer.exe의 하위 프로세스에 위치하였다(그림 3 참조). 그러나 Windows 10 환경에서 UWP(Universal Windows Platform) 앱의 형태로 변경됨에 따라 svchost.exe의 하위 프로세스로 실행된다. 이러한 차이점에 착안하여 계산기를 희생자 프로세스로 선택했다. 마지막으로 CCS는 Texas Instruments(TI)사의 통합 개발 환경(IDE)으로, 산업제어시스템을 위한 임베디드 애플리케이션 개발에 많이 사용되고 있다. 최근 IoT(Internet of Things) 및 스마트 팩토리 산업 등이 발전하게 되면서 산업제어시스템을 대상으로 한 사이버 공격이 증가하면서, 임베디드 애플리케이션을 개발하는 통합 개발 환경도 공격자들의 대상이 되고 있다. 따라서 IDE가 인젝션 공격에 얼마나 강인한 가를 평가하기 위한 예로 CCS를 희생자 프로세스에 포함한다.

4.3 인젝션 공격 실험

네 개의 희생자 프로세스들을 대상으로, 표 1의 인젝션 공격 실험을 수행한다. Process Explorer로 해당 희생자 프로세스의 PID와 TID 정보를 확인하고, 획득한 정보를 Pinjectra 실행 시 명령 옵션으로 그림 4와 같이 Demo ID와 함께 입력하여 실행한다.

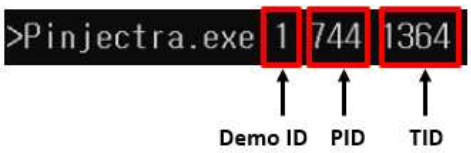

그림 4. Pinjectra 실행 명령
Fig 4. Pinjectra execution command

이렇게 실행된 프로세스는 Demo ID 별로 공격 성공 여부를 표시해 주는 메시지 박스나 메시지를 출력하게 된다. 그림 5는 공격이 성공되어

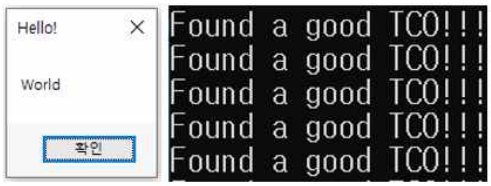

그림 5. 공격 성공 시 나타나는 메시지
Fig 5. Messages displayed when an attack is succeeded

나타나는 메시지 박스(메시지)의 예를 보여준다. 이러한 방식으로 4가지 희생자 프로세스들에 대한 12가지 인젝션 공격 실험 결과를 표 3에 작성하였다.

표 3에서 공격이 성공한 경우, 즉, 공격을 시도하였을 때 메시지나 메시지 박스가 발생한 경우에는 “O”를 표시하고, 아닌 경우에는 “X”로 표시한다. 동일한 공격을 여러 번 시도하였을 때, 공격이 실패한 경우가 많지만 가끔 공격이 성공한 경우가 나타나면 “O/X”로 표시한다. 이 외에 “Pinjectra Error”로 표시된 것은, 그림 6과 같이 공격을 시도했을 때, Pinjectra가 WerFault.exe와 함께 종료되는 현상을 말한다. “Process Error”는, 그림 7과 같이 희생자 프로세스가 WerFault.exe 와 함께 종료되는 현상을 나타낸다. 참고로 WerFault.exe는 Windows 오류 보고(Windows Error Reporting)를 위해 사용되는 서비스로, 이를 사용하여 마이크로소프트는 OS, Windows 기능, 애플리케이션에 관련된 오류를 추적하고 해결한다.

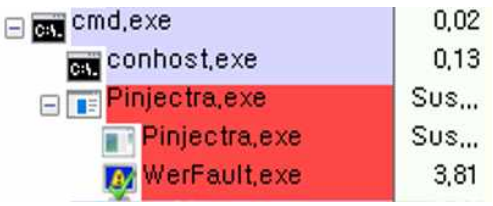

그림 6. “Pinjectra Error”시 출력 화면
Fig 6. Output screen in case of “Pinjectra Error”

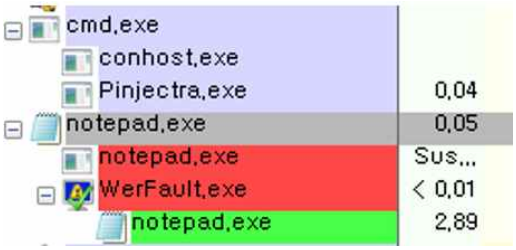

그림 7. “Process Error”시 출력 화면
Fig 7. Output screen in case of “Process Error”

4.4 결과 분석

표 3을 보면, 전체적으로 희생자 프로세스들에서 인젝션 공격이 성공하고 실패하는 패턴이 유사함을 확인할 수 있다. Demo ID 기준으로, 공격이 모두 실패한 기법은 2, 6번이였고, 5번과 11번에 대해서도 “Pinjectra Error” 또는 “Process Error”를 발생시키며 성공하지 못하였다. 6번의 경우는 콘솔 프로그램에 대한 기법이기 때문에, 4가지 희생자 프로세스에 대해서 모두 공격이 실패하는 것으로 판단된다.

계산기의 경우 12가지 공격 기법 중 총 3가지(7, 8, 10번)에 대해서만 성공한 결과를 보이며, 다른 희생자 프로세스와는 다르게 절반 이상의 공격 기법들이 실패하였다. 이는 현재 Windows 10 환경에서 사용되는 계산기가 UWP 기반의 앱 형태로 배포되고 있기 때문이다. UWP는 샌드박싱을 사용하므로 앱이 불필요한 리소스에 대해 액세스할 수 없다. 따라서 악성 애플리케이션이 하드웨어 주소를 직접 지정하거나 디바이스 드라이버를 설치 혹은 핵심 운영체제 요소를 수정하는 것을 방지할 수 있다[34]. 이처럼 Windows 10에서의 UWP 앱은 인젝션 공격 면에서 기존의 Win32 프로그램과 상당한 차이를 보여준다.

Demo ID 8, 10번 공격은 파일 탐색기(Explorer.exe)에 최적화된 공격으로, 파일 탐색기만 실행되고 있다면 언제든지 가능한 공격이다. 파일 탐색기는 Windows 환경에서의 GUI Shell로 GUI를 제공하여 파일 시스템, 작업표시줄, 바탕화면 등 모니터상의 많은 사용자 인터페이스 항목을 표현하는 운영체제의 구성요소이다. 이에 따라, 해당 공격 기법이 계산기에 대해서 성공하게 된다.

표 3. 네 개의 응용 프로그램을 대상으로 한 인젝션 공격 실험 결과
Table 3. Results of injection attack experiments with four application programs

Demo ID	Calculator		Notepad		Code Composer Studio		Explorer	
	1903	2004	1903	2004	1903	2004	1903	2004
T1	X	X	O	O	O	O	O	X
T2	X	X	X	X	X	X	X	X
T3	X	X	O	O	O	O	O	O
T4	X	X	O / X	O	O	O	O	O
T5	Pinjectra Error	Pinjectra Error	Pinjectra Error	Pinjectra Error	Pinjectra Error	Pinjectra Error	Pinjectra Error	Pinjectra Error
T6	X	X	X	X	X	X	X	X
T7	O	O	O	O	X	X	O	O
T8	O	O	O	O	O	O	O	O
T9	X	X	O	X	O / X	X	X	X
T10	O	O	O	O	O	O	O	O
T11	Process Error	Process Error	Process Error	Process Error	Process Error	Process Error	Process Error	Process Error
T12	X	X	O	O	O	O	O	O

Demo ID 1, 3, 4번 공격은 매우 오래된 기법임에도 불구하고, 계산기를 제외한 모든 희생자 프로세스에 대해 공격이 성공하였다. 계산기가 다른 희생자 프로세스에 비해 인젝션 공격에 강인함을 알 수 있다. 표 3을 보면 CCS에 대한 인젝션 공격 성공률이 낮지 않아, 임베디드 애플리케이션을 대상으로 한 인젝션 위협이 심각함을 확인할 수 있다.

5. 인젝션 공격에 대한 완화 기술의 효용성 분석

본 절에서는 인젝션 공격을 완화하기 위해 마이크로소프트사에서 제공하는 방어 기술을 적용해본다.

성공한 인젝션 공격 기법들을 대상으로 2.2절에서 언급한 완화 기술들의 효과를 분석한다. 3가지 완화 기술들, 즉 CFG. CIG, ACG를 희생자 프로세스에 적용시킨 후 인젝션 공격을 시도하고 결과를 확인한다.

완화 기술들을 적용하기에 앞서, [4]에서 평가한 각 실행 기법, 쓰기 기법들의 평가 기준 중 CFG/CIG에 대해 정리한 내용이 표 4에 나타나 있다.

CFG/CIG 방어 기술들의 효과를 검증하기 위해, 표 3과 표 4를 비교하면서 표 3에서 공격이 성공된 인젝션 기법들에 대해 관련성 있는 방어 기술을 표 4에서 찾아 적용한다. 표 3을 보면, 메모장에 대해 Demo ID 1, 3, 4, 7, 8, 10, 12번 공격 기법이 성공하였다. 이를 표 4와 비교하여, Demo ID 1, 3, 7, 8, 10번 기법에 대해 완화 기술을 적용할 수 있다.

메모장을 대상으로 완화 기술을 적용하고, 완화 기술이 잘 적용되었는지 확인한 결과가 그림 8에 나타나 있다. CIG 정책 적용을 나타내는

표 4. [4]에서 정리한 CFG/CIG의 효능
Table 4. Effectiveness of CFG/CIG in [4]

Demo ID	CFG/CIG
1	CIG : Microsoft가 서명하지 않은 DLL 로드를 방지
2	CIG : Microsoft가 서명하지 않은 DLL 로드를 방지
3	Target : CFG-valid
4	-
5	Target : CFG-valid
6	-
7	Target : CFG-valid
8	Target : CFG-valid
9	-
10	Target : CFG-valid
11	-
12	-

MicrosoftSignedOnly가 ON되어 있고, ACG에 해당하는 BlockDynamicCode가 ON되어 있는 것을 확인할 수 있다. CFG는 컴파일 시점에서 Visual Studio 옵션 설정에 의해 적용 여부가 결정된다. CFG 설정 여부를 확인하기 위해, 그림 9와 같이 dumpbin[36]을 사용하여 CFG 적용 여부를 확인하였다. 참고로 CFG를 우회하는 기법이 [4]에서 소개되었으나, 이는 1809 버전 이후로 마이크로소프트 보안 패치에 의해 작동이 중지되었다.

그림 9와 같이 “Guard CF function count”에 할당된 16진수 값 “7D”를 통해 CFG 적용 여부를 확인하고, 4장과 같이 실험을 수행하였다. Demo ID 1, 3, 7, 8, 10번 기법들을 실험한 결과, 1번에 대해서 오류가 발생하였다. 즉, 그림 10과 같이 CIG를 적용하여 마이크로소프트사의 시그니처가 존재하지 않는 DLL을 로드하는 시도가

실패하였다. 4가지 희생자 프로세스를 대상으로 실험해본 결과, 표 3의 Demo ID - T1의 경우 메모장을 대상으로 CIG 기술을 적용했을 때를 제외하고 모두 방어에 실패하는 것을 확인할 수 있었다.

```
ProcessName                    : notepad.exe
Source                         : Registry
Id                             : 0
CFG:
    Enable                             : ON
    SuppressExports                    : ON
BinarySignature:
    MicrosoftSignedOnly                : ON
    AllowStoreSignedBinaries           : ON
DynamicCode:
    BlockDynamicCode                   : ON
```

그림 8. 메모장에 완화 기술을 적용한 결과
Fig 8. Result of applying mitigation techniques to Notepad

```
004015BC Guard CF function table
      7D Guard CF function count
00017500 Guard Flags
```

그림 9. 메모장에서 CFG 적용 결과
Fig 9. Result of applying the CFG to Notepad

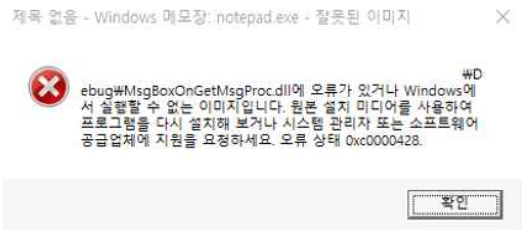

그림 10. CIG가 비인가 DLL 적재를 방지
Fig 10. CIG stops loading unsigned DLL

6. 결 론

MS Windows 운영체제(OS)에서 프로세스 인젝션, DLL 인젝션, 코드 인젝션 등의 인젝션 공격이 여전히 중요 보안 위협이 되고 있다. 본 논문에서는 인젝션 공격의 유형을 분석하고, 계산기, 메모장, CCS, 파일 탐색기 프로그램들을 대상으로 12가지 인젝션 공격을 적용해 보았다. 64비트 Windows 10 OS의 두 버전에서, Pinjectra를 사용하여 인젝션 공격 실험을 수행한 결과를 분석하였다. 실험 결과, 마이크로소프트사의 인젝션 공격 방어 기술로 인해 보완된 사항도 있지만, 일부 프로그램에 대해서는 전통적인 인젝션 공격들을 물론이고 최신 인젝션 공격도 적용 가능하였다. 따라서 프로세스 인젝션 공격에 대한 방어 기법을 개선하는 연구가 필요함을 파악할 수 있었다.

본 논문에서는, 인젝션 공격을 탐지하고 방어하는 기존 연구들로, 해시를 사용한 메모리 영역 비교, API 후킹, 포렌식을 통한 데이터 기반 블랙리스트 구축 기법 등도 정리하였다.

최근에는 IoT 및 스마트 팩토리 산업이 활성화되면서 산업제어시스템이 사이버 공격의 대상이 되고 있다. 향후에는, 실험 결과로 작성한 4.3절 표 3에서 “O/X”로 표기한 부분과 4.4절의 내용 중 2번 기법이 실패한 이유에 대해 구체적으로 분석할 예정이다. 또한, 임베디드 애플리케이션 통합 개발 환경인 CCS(Code Composer Studio)의 실험 결과를 바탕으로 인젝션 공격을 효과적으로 탐지하고 방어하는 연구를 심층적으로 수행할 계획이다.

본 연구는 산업통상자원부(MOTIE)와 한국에너지기술평가원(KETEP)의 지원을 받아 수행한 연구과제임.
(NO. 20171510102080)

참 고 문 헌

[1] 위키백과, https://ko.wikipedia.org/wiki/Microsoft_Windows

[2] 황현욱; 채종호; 윤영태. 윈도우 환경에서의 메모리 인젝션 기술과 인젝션 된 DLL 분석 기술. 융합보안논문지, 2006, 6.3: 59-67. UCI : G704-001662.2006.6.3.004

[3] C. S. Wright, “Taking control, Functions to DLL injection”, March 2007. https://dx.doi.org/10.2139/ssrn.3153492

[4] Amit Klein, Itzik Kotler, “Windows Process Injection in 2019”, Black Hat USA 2019, 2019. https://i.blackhat.com/USA-19/Thursday/us-19-Kotler-Process-Injection-Techniques-Gotta-Catch-Them-All-wp.pdf

[5] Hosseini, Ashkan, “Ten Process Injection Techniques: A Technical Survey of Common and Trending Process Injection Techniques”, Endpoint Security Blog (2017). https://www.elastic.co/kr/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process

[6] 보안뉴스, 파일리스 위협과 랜섬웨어의 결합으로 탄생한 멀웨어 등장, 2017.06, Available at: https://www.boannews.com/media/view.asp?idx=55391&page=1&kind=3

[7] 보안뉴스, 넷워커 랜섬웨어, 사업 모델 바꾸더니 순식간에 수익 불어나, 2020.08, Available at: https://www.boannews.com/media/view.asp?idx=90291&page=1&kind=1

[8] 보안뉴스, 2020년 1분기 최악의 신규 랜섬웨어 5종 꼽아보니... ‘코로나’ 키워드 악용, 2020.06, Available at: https://www.boannews.com/media/view.asp?idx=89122&page=1&kind=1

[9] A. H. A. Kamal et al., “Cybersecurity Issues and Challenges during Covid-19 Pandemic”, Preprints, 2020. 2020090249. https://doi.org/10.20944/preprints202009.0249.v1

[10] S. Fewer, “Reflective DLL injection”, Harmony Security, Version 1, 2008 https://github.com/stephenfewer/ReflectiveDLLInjection

[11] M. Gorelik and R. Moshailov, “Fileless Malware: Attack Trend Exposed”, Morphisec Ltd, 2017. https://blog.morphisec.com/fileless-malware-attack-trend-exposed

[12] B. L. Krishna, “Comparative Study of Fileless Ransomware”, International Journal of Trend in Scientific Research and Development (IJTSRD), 4(3): pp. 608-616, April 2020. https://www.ijtsrd.com/engineering/computer-engineering/30600/comparative-study-of-fileless-ransomware/krishna-b-l

[13] K. McCammon, et al., “2020 Threat Detection Report”, Red Canary: Improve Security with Threat Detection, 2020.03 https://redcanary.com/threat-detection-report/introduction/

[14] MITRE ATT&CK®, “Process Injection”, https://attack.mitre.org/techniques/T1055/

[15] https://github.com/SafeBreach-Labs/pinjectra

[16] Microsoft Docs., “Process Explorer v16.32” https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer

[17] Userland API Monitoring and Code Injection Detection, https://0x00sec.org/t/userland-api-monitoring-and-code-injection-detection/5565

[18] Kaspersky Lab., https://encyclopedia.kaspersky.com/glossary/code-injection/

[19] Kaspersky Lab., https://encyclopedia.kaspersky.com/glossary/dll-injection/

[20] https://attack.mitre.org/techniques/T1055/008/

[21] https://attack.mitre.org/techniques/T1055/009/

[22] S. Sayeed, et al., “Control-flow integrity: Attacks and protections”, Applied Sciences 9.20 (2019): 4229. https://doi.org/10.3390/app9204229

[23] Microsoft Docs., “Control Flow Guard”,

https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard
[24] Z. Yunhai, “Bypass control flow guard comprehensively”, Black Hat USA (2015). https://www.blackhat.com/docs/us-15/materials/us-15-Zhang-Bypass-Control-Flow-Guard-Comprehensively-wp.pdf
[25] Weston, David, and Matt Miller, “Microsoft’s strategy and technology improvements toward mitigating arbitrary native code execution”, CanSecWest 2017 (2017). https://cansecwest.com/slides/2017/CSW2017_Weston-Miller_Mitigating_Native_Remote_Code_Execution.pdf
[26] 임수민; 임을규, 프로세스 가상 메모리 데이터 유사성을 이용한 프로세스 할로윙 공격 탐지, 정보보호학회논문지, 2019, 29.2:431-438. https://doi.org/10.13089/JKIISC.2019.29.2.431
[27] Github Repository, “Captain”, https://github.com/y3n11/Captain
[28] Github Repository, “UnRunPE”, https://github.com/NtRaiseHardError/UnRunPE
[29] Github Repository, “Dreadnought”, https://github.com/NtRaiseHardError/Dreadnought
[30] Github Repository, “Rekall discontinuation”, https://github.com/google/rekall
[31] Github, “Volatility Foundation”, https://github.com/volatilityfoundation
[32] SRIVASTAVA, Anurag; JONES, James H., Detecting code injection by cross-validating stack and VAD information in windo ws physical memory, In: 2017 IEEE Confe rence on Open Systems (ICOS). IEEE, 20 17. pp. 83-89. https://doi.org/10.1109/ICOS.2017.8280279
[33] Balaoura, Sotiria, “Process injection techniques and detection using the Volatility Framework”, MS thesis, University of Piraeus, 2018. http://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/11578/Balaoura_MTE1623.pdf?sequence=1&isAllowed=y
[34] BLOCK, Frank; DEWALD, Andreas, Windows Memory Forensics: Detecting (Un) Intentionally Hidden Injected Code by Examining Page Table Entries, Digital Investigation, 2019, 29: S3-S12. https://doi.org/10.1016/j.diin.2019.04.008
[35] InfoWorld, “Microsoft UWP boosts security for Windows apps”, https://www.infoworld.com/article/3049955/microsoft-uwp-boosts-security-for-windows-apps.html
[36] Microsoft Docs., “DUMPBIN Reference”, https://docs.microsoft.com/en-us/cpp/build/reference/dumpbin-reference?view=msvc-160

저 자 소 개

성호준(HoJun Seong)

2015.3-현재 단국대학교 응용컴퓨터공학과 학사과정 재학
<주관심분야> 시스템 보안, 임베디드 보안

이호웅(HoWoong Lee)

1998년 호서대학교 컴퓨터공학과 공학사
2000년 인하대학교 전자계산공학과 공학석사
2000년~2020년 2월 ㈜안랩 CTO
2020년 3월~현재 호서대학교 컴퓨터정보공학부 교수
<주관심분야> 인공지능, 컴퓨터보안, ICS 보안, 디지털 헬스케어, 블록체인 등

조창연(ChangYeon Cho)

2014.3-현재 단국대학교 소프트웨어학과 학사과정 재학
<주관심분야> 시스템 보안, 네트워크 보안

조성제(Seong-je Cho)

1989년 서울대학교 컴퓨터공학과 공학사
1991년 서울대학교 컴퓨터공학과 공학석사
1996년 서울대학교 컴퓨터공학과 공학박사
2001년 미국 University of California, Irvine 객원 연구원
2009년 미국 University of Cincinnati 객원 연구원
1997년 3월~현재 단국대학교 소프트웨어학과/컴퓨터학과 교수
<주관심분야> 컴퓨터보안(스마트폰보안), 소프트웨어 지적재산권 보호, 시스템 소프트웨어 등