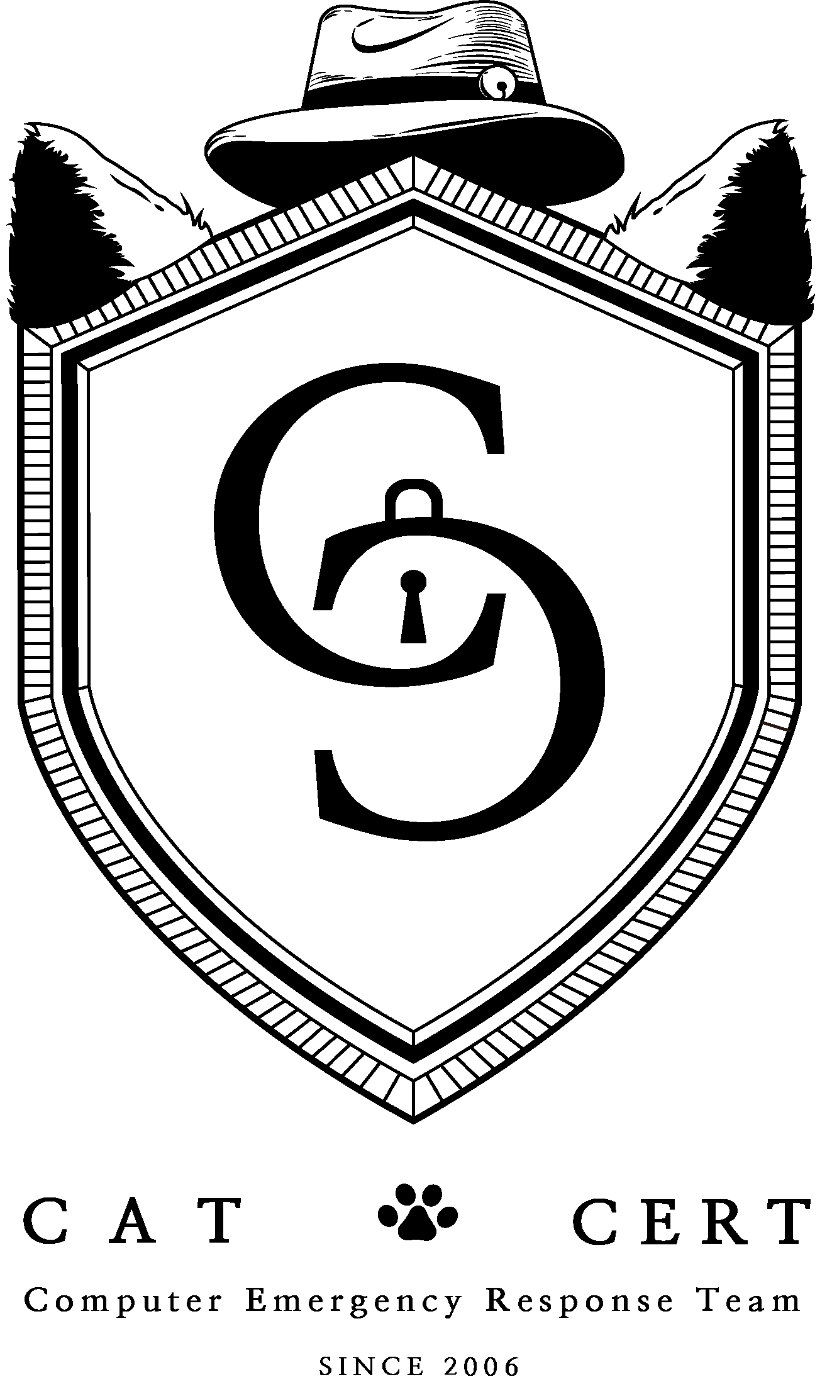
**캣서트 보고서**



202221649 인공지능학과 김민영

REV\_study2

3.

**PickMe.exe**

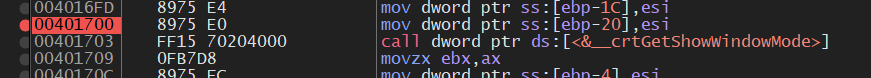


초반에 함수들을 살펴보며 안티디버깅이 사용되었나 확인해보았음

함수를 지나가도 디버깅이 진행되며 프로그램 정상 작동됨을 확인 -> 안티디버깅 사용되지 않았다고 생각

텍스트이(가) 표시된 사진

자동 생성된 설명



윈도우 창을 띄워주는 것처럼 생긴 함수들에 bp를 걸어놓고 한줄씩 실행시켜보며 어디서 프로그램 창이 실행되는지 확인해보았음

텍스트이(가) 표시된 사진

자동 생성된 설명

ShowWindow함수로 창을 띄워준다는 것을 확인 / time과 srand 함수를 이용해 srand(time(0))로 그림을 무작위로 출력시켜주게 하는 코드 부분이라고 생각했음

**안티디버깅 종류 조사**

안티디버깅이란?

디버깅을 방해해서 분석을 어렵게 하는 기술 / 디버깅을 당한다면 디버깅을 하지 못하도록 고의적으로 에러를 발생 or 디버거 프로그램을 종료시키는 등 다양한 방법 사용해서 방해함

크게 static 그룹과 dynamic 그룹으로 나뉨

1. **Static**

* 디버거를 탐지하여 프로그램이 정상적으로 실행되지 못하게 함
* 디버깅을 시작할 때 한 번만 해체해주면 됨

#BeingDebugged

PEB – (프로세스 정보를 담고 있는 구조체)값을 참조해서 할 수 있는 방식

텍스트이(가) 표시된 사진

자동 생성된 설명

디버깅 중일 때는 1(True)로 설정 / 디버깅 중이 아닐 때는 0 (False)로 설정

#Ldr

PEB.Ldr에 있는 Ldr 구조체의 주소를 가리키고 있는 값을 참조

텍스트이(가) 표시된 사진

자동 생성된 설명

LDR – 프로세스가 디버깅 시에 메모리의 힙 영역에 디버깅 당하는 표시를 메모리를 해줌

프로세스가 디버깅 당하면 힙 영역에 0xFEEEFEEE 또는 0xABABABAB 를 채움

→ 해당 표시로 디버깅 중임을 확인

+ PEB – 프로세스 정보를 담고 있는 구조체

#TLS 콜백 함수 **/ IsDebuggerPresent**

TLS 콜백 함수 – 프로세스의 스레드가 생성 및 종료 될 때마다 자동으로 호출되는 콜백 함수

코드의 시작점인 EP 코드보다 먼저 호출됨을 이용(EP 코드 실행 전에 TLS 콜백 함수가 실행됨)

– 브레이크 포인트 사용함

ISDebuggerPresent 함수를 이용함 / 현재 프로세스가 디버깅중이면 “Debugger Detected!” 문자열을 출력함

커널 모드의 디버거는 탐지 X, 유저 모드의 디버거만 탐지 O

#NTQueryObject

디버깅 중면 DebugObject 라는 객체를 생성함 – 객체 탐지를 위해 NTQueryObject 함수를 이용함

DebugObject의 리스트를 나열해서 존재하는 객체의 개수를 카운팅함, + 다양한 커널 객체 정보를 구해옴

객체의 개수가 1개 – 디버거가 발견 X

#NTQueryFlag

PEB 구조체의 0x68 에 존재함

디버깅 중에 0x70 으로 세팅됨 / 정상일 때는 0으로 세팅

1. **Dynamic**

* 디버거 트레이싱을 방해해서 원본 프로그램의 동작 원리를 이해하는데 방해함
* 정상적 분석을 위해서는 dynamic 기법의 안티디버깅을 만날 때마다 해체 해야함

#Timing Check

특정 명령어 사이의 시간 간격을 측정함

→ 디버깅 시에 코드를 한 줄씩 실행하기에 실행할 때 시간이 많이 들어감

Counter Based Metod / Timed Based 방식으로 나뉨

* Counter Based Metod

CPU의 카운터를 이용함

* Timed Based

시스템의 실제 시간을 이용함

#0x CC Detection

디버깅 시에 BreakPoint 를 설치하게 되면 해당 위치 코드가 0x CC로 변경됨

API BreakPoint / CheckSum 방식으로 나뉨

* API BreakPoint

리버서들은 특정 API 에 중단점을 설치하고 디버깅을 진행하곤 함 → 이를 역이용하여 API 코드 시작 부근에 0x CC 가 있는지 확인함

* CheckSum

특정 코드 영역의 CheckSum 값을 구하고 미리 계산해놓은 원본 CheckSum과 비교함

0x CC 로 변경되는지 확인

#SEH 메커니즘 / Exception

정상적 프로세스에서 예외(Exception)가 발생하면 OS에서 예외를 받음 → 프로세스에 등록된 SEH를 호출해줌

예외 : 하드웨어 예외 (하드웨어에서 인식하고 알려줌) / 소프트웨어 예외 (소프트웨어에서 감지)

**#`NTQueryInformationProcess()`를 이용한 안티디버깅 구현 및 우회**

typedef DWORD(WINAPI\* APINtQueryInformationProcess) (HANDLE, DWORD, PVOID, DWORD, PVOID);

NtQueryInformationProcess를 정의, 인자를 받아줌

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

**#패킹 / 언패킹 조사**

**패킹 :** 실행 압축

실행 파일 내부에 있는 코드를 압축하여 평소에는 코드를 압축한 상태로 저장

→ 파일을 실행하면 메모리에서 압축을 해제 → 파일을 실행

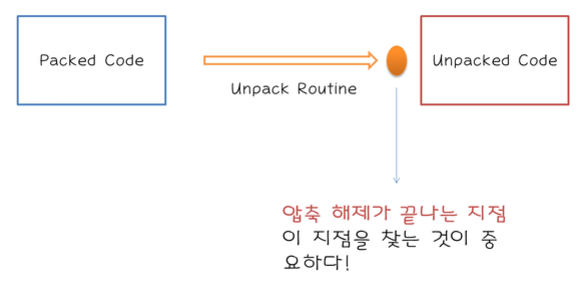
Zip 파일처럼 압축을 해제해야 프로그램을 실행하는 방식 X

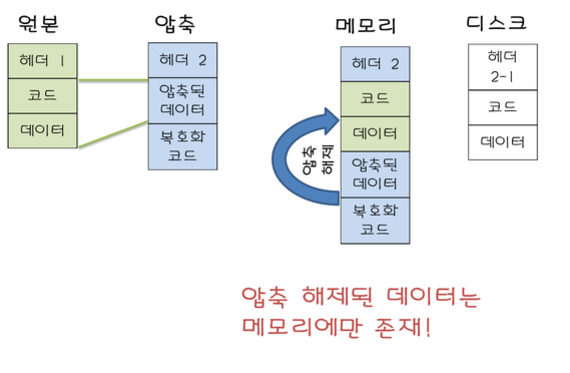
일반 프로그램처럼 바로 실행 가능 O

<WHY?>

파일 압축시 용량을 더 적게 사용 – 프로그램의 크기를 줄임(전 후 3~4배의 차이)

→ 작은 용량으로 악성코드를 빠르게 전파할 수 있음





**언패킹 :** 패킹했던 파일을 해체함

<기본 원리> - 압축이 해제된 상태의 코드를 dump 함

실행 압축 프로그램이 실행되는 시점에서는 압축되기 전 상태로 돌아가야함

그때의 메모리 상태를 메모리 내용 그대로 파일에 저장함 → Entry Point 진입점을 파일을 압축하기 전 위치로 설정해줌

언패킹 루틴은 마지막에 EP로 점프하여 원본코드가 적상작동할 조건을 만들어줌

OEP로 순서가 달라지는 부분을 찾음 → 메모리에 위치하는 언패킹된 데이터를 얻을 수 있음

언패킹된 데이터를 덤핑 = 원본코드를 얻음!

+ Dump : 장애 발생시 그 상태를 출력, 인쇄하기 위해 기억장치나 파일 내용, 자료에 기억시킨 내용을 보조기억장치 등에 복사 or 전이하는 조작 / 컴퓨터 주기억장치나 레지스터 등에 해당 내용들이 존재 및 지정된 메모리 영역