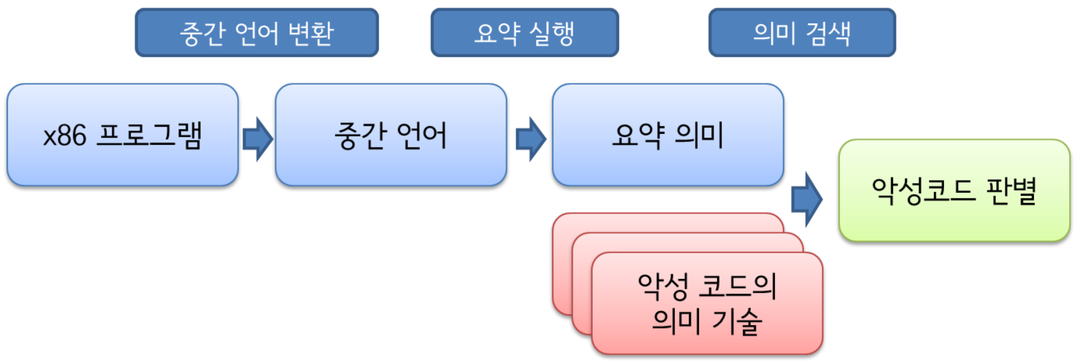
x86 실행파일에서 악성코드의 행동을 정적으로 분석하기

이승중
서울대학교 프로그래밍 연구실

의미기반의 정적 분석으로 악성코드의 행동 검출



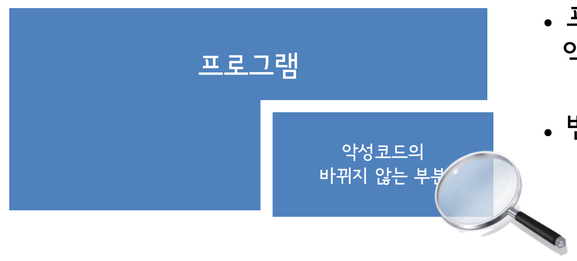
실행파일이 악성코드의 행동을 하는지 여부를
의미 기반의 **정적 분석**을 이용하여 검출한다.

이진 프로그램과 악성 행동 의미를 기술한 언어의 리스트를 입력으로 받아
프로그램이 나쁜 행동을 하는지 여부를 알려주는 분석기

- 의미 기반: 프로그램의 모양새만을 보지 않는 분석
- 정적 분석: 직접 실행시켜보지 않고 미리 모든 실행경로를 아우르는 분석

기존 악성코드 탐지의 문제점

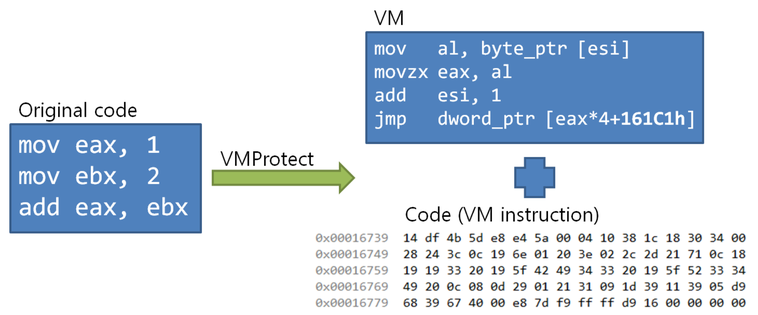
기존 악성코드 탐지방법 시그니처 기반



- 프로그램의 실행코드/데이터에서
  악성코드 고유의 모양을 살펴봄
- 변종이 나올 때마다 데이터베이스 업데이트 필요

VM obfuscation 적용된 악성코드의 출현

매번 다른 obfuscation을 적용하여 모양이 달라짐



x86 중간언어의 요약 실행

프로그램의 정의

$$
\begin{array}{rcrl}
pgm & \in & Program & = (Addr \times Byte)^* \times Addr \\
a & \in & Addr & = Section \times \mathbb{Z} \\
s & \in & Section & = \mathbb{Z}
\end{array}
$$

$$
\begin{array}{rcll}
C & ::= & \texttt{mov}_n\ E\ E & \\
 & | & \texttt{jmpnz}\ E\ E & \text{// conditional jump} \\
 & | & C;C & \text{// sequence} \\
 & & & \\
E & ::= & x & \text{// variable (register, flag)} \\
 & | & \texttt{new} & \text{// allocate new memory section} \\
 & | & n & \text{// integer, address} \\
 & | & [E]_n & \text{// load memory} \\
 & | & \star\ E & \text{// unary operator} \\
 & | & E \diamond E & \text{// binary operator}
\end{array}
$$

요약 도메인

$$
\begin{array}{rcrl}
(\hat{\sigma},\ \hat{m},\ \hat{ip})\ \text{as}\ \hat{st} \in & \hat{State} & = & \hat{Env} \times \hat{Mem} \times \hat{Addr} \\
\hat{\sigma} \in & \hat{Env} & = & Var \xrightarrow{\text{fin}} \hat{Value} \\
\hat{v} \in & \hat{Value} & = & (\mathbb{Z}_\perp + \hat{Addr}) \times \hat{Addr} \\
\hat{m} \in & \hat{Mem} & = & Addr \xrightarrow{\text{fin}} (2^{Byte \times \mathbb{Z}} + \hat{Addr}) \times \hat{Addr} \\
 & \hat{Addr} & = & 2^{Addr} \quad \text{(set of program locations)} \\
 & \hat{Trace} & = & 2^{\hat{State}}
\end{array}
$$

요약 실행

$$[\![\cdot]\!] : Program \to \hat{Trace}$$

$$[\![(a_1,\ b_1)(\hat{a_2},\ b_2)...,a]\!] = fix(\hat{F} \stackrel{\text{def}}{=} \lambda\hat{T}.\{\hat{st_0}\} \cup \hat{Next}\ \hat{T})$$

$$\hat{st_0} = \langle\ \{\}, \{a_1 \mapsto \{(b_1,\ 0)\},\ a_2 \mapsto \{(b_2,\ 0)\}, ...\ \},\ \{a\} \rangle$$

$$
\begin{array}{rcl}
\hat{Next} & = & (\rho \sqcup) \circ \hat{\pi} \circ (\rho_\cup\ \hat{next}) \\
\hat{next} & = & \lambda\hat{st}.\{\hat{st'} | \hat{st} \hat{\to} \hat{st'}\}
\end{array}
$$

x86 바이너리 프로그램의 요약 실행은

프로그램이 실제로 실행되었을 때

프로그램 카운터가 가리키는 모든 지점에서의

메모리와 레지스터의 요약된 상태들의 집합을 얻는 것

악성코드의 행동 표현

행동을 기술하는 언어

$$
\begin{array}{rcll}
c & := & b & \\
 & | & b \wedge b & \\
 & & & \\
b & := & \mathit{InsideCodeSection}(e) & \text{address belongs to code section} \\
 & | & \{e\ (e^*)\} & \text{function call is in the code} \\
 & | & \{e := e\} & \text{assignment is in the code} \\
 & | & \{\mathit{x86\_instruction}\} & \text{x86 instruction is in the code} \\
 & | & \{\mathit{x86\_instruction}\}\ \text{with}\ c & \text{x86 instruction is in the code with given condition} \\
 & | & e = e & \text{values of two expressions are equal} \\
 & | & \neg b & \\
 & | & b \vee b &
\end{array}
$$

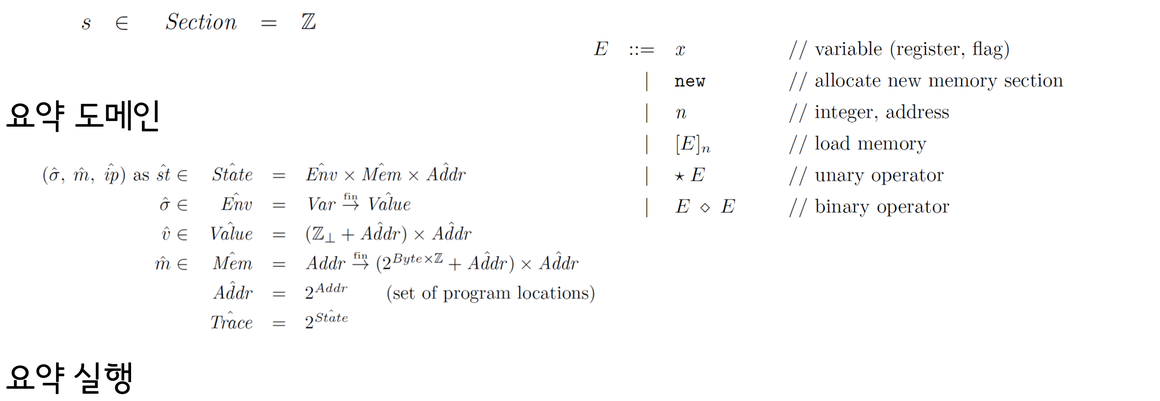
$$
\begin{array}{rcll}
e & := & x & \text{variable} \\
 & | & \texttt{r} & \text{register(predefined variable)} \\
 & | & \texttt{f} & \text{external function, variable} \\
 & | & \_ & \text{don't care} \\
 & | & n & \text{integer constant} \\
 & | & s & \text{string} \\
 & | & e\ (e^*) & \text{function call} \\
 & | & e[e] & \text{array} \\
 & | & \circ e \mid e \star e & \text{unary/binary operator}
\end{array}
$$

시스템 행동 가로채기의 표현

SDT 가로채기	$x = \texttt{KeServiceDescriptorTable}$ $\wedge\ y = \texttt{MmCreateMdl}(\_,\ x,\ \_,\ \_)$ $\wedge\ z = \texttt{MmMapLockedPages}(\_,\ y)$ $\wedge\ z[\_] = f$ $\wedge\ \mathit{InsideCodeSection}(f)$
WriteProcessMemory 진입점 가로채기	$y = \texttt{GetProcessAddress}(\_, x)$ $\wedge\ (x = \mathit{String\ of\ system\text{-}call\ function})$ $\wedge\ \neg \mathit{InsideCodeSection}(y)$ $\wedge\ \{\texttt{WriteProcessMemory}(\_,\ y,\ \_,\ \_)\}$
IDT 가로채기	$\{\texttt{sidt}\ x\}$ $\wedge\ y = (x[5]\ <<24 \mid x[4]\ <<\ 16 \mid x[3]\ <<8 \mid x[2])$ $\wedge\ y[\_] = z$ $\wedge\ \mathit{InsideCodeSection}(z)$

악성코드의 주요 동작 시스템 행동 가로채기(hooking)

SDT(Service Description Table) 가로채기



SDT 가로채기를 수행하는 실제 C 코드

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT theDriverObject, IN PUNICODE_STRING theRegistryPath)
{
   ...
   // save old system call locations
   OldZwQuerySystemInformation =(ZWQUERYSYSTEMINFORMATION)(SYSTEMSERVICE(ZwQuerySystemInformation));

   // Map the memory into our domain so we can change the permissions on the MDL
   g_pmdlSystemCall = MmCreateMdl(NULL,
                                  KeServiceDescriptorTable.ServiceTableBase,
                                  KeServiceDescriptorTable.NumberOfServices*4);

   if(!g_pmdlSystemCall)
      return STATUS_UNSUCCESSFUL;

   MmBuildMdlForNonPagedPool(g_pmdlSystemCall);

   // Change the flags of the MDL
   g_pmdlSystemCall->MdlFlags = g_pmdlSystemCall->MdlFlags | MDL_MAPPED_TO_SYSTEM_VA;

   MappedSystemCallTable = MmMapLockedPages(g_pmdlSystemCall, KernelMode);

   // hook system calls
   HOOK_SYSCALL( ZwQuerySystemInformation, NewZwQuerySystemInformation, OldZwQuerySystemInformation );

   return STATUS_SUCCESS;
}
```

여러 가지 가로채기 방법들

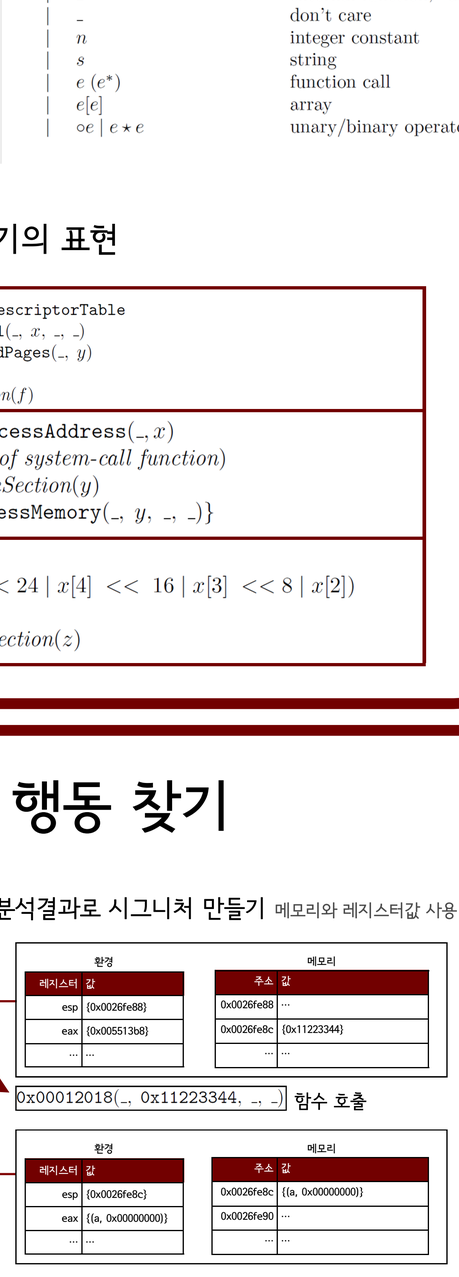
- IDT(Interrupt Descriptor Table) 가로채기
- 커널 함수 진입부 덮어씌우기
- sysenter 명령어 가로채기
- 시스템 콜 진입점 가로채기
- FltRegisterFilter 함수 사용
- WriteProcessMemory 함수 사용

분석결과에서 악성코드의 행동 찾기

프로그램 sdt 가로채기

```
...
000111C8: 51              push    ecx
000111C9: FF 30           push    dword ptr [eax]
000111CB: 57              push    edi
000111CC: FF 15 18 20 01 00  call    dword ptr [__imp__MmCreateMdl@12]
000111D2: 3B C7           cmp     eax,edi
000111D4: A3 10 30 01 00  mov     dword ptr [_g_pmdlSystemCall],eax
000111D9: 75 07           jne     000111E2
000111DB: B8 01 00 00 C0  mov     eax,0C0000001h
000111E0: EB 37           jmp     00011219
000111E2: 50              push    eax
000111E3: FF 15 14 20 01 00  call    dword ptr [__imp__MmBuildMdlForNonPagedPool@4]
000111E9: A1 10 30 01 00  mov     eax,dword ptr [_g_pmdlSystemCall]
000111EE: 80 48 06 01     or      byte ptr [eax+6],1
000111F2: 57              push    edi
000111F3: FF 35 10 30 01 00  push    dword ptr [_g_pmdlSystemCall]
000111F9: FF 15 10 20 01 00  call    dword ptr [__imp__MmMapLockedPages@8]
000111FF: A3 24 30 01 00  mov     dword ptr [_MappedSystemCallTable],eax
00011204: 8B 56 01        mov     edx,dword ptr [esi+1]
00011207: B9 0E 10 01 00  mov     ecx,offset _NewZwQuerySystemInformation@4
0001120C: 8D 04 90        lea     eax,[eax+edx*4]
0001120F: 87 08           xchg    ecx,dword ptr [eax]
00011211: 89 0D 20 30 01 00  mov     dword ptr [_OldZwQuerySystemInformation],ecx
```

분석결과로 시그니처 만들기 메모리와 레지스터값 사용



프로그램의 정보를 이용하여 행동의 표현을 줄임

$$
\begin{array}{ll}
 & x = \texttt{KeServiceDescriptorTable} \\
\wedge & y = \texttt{MmCreateMdl}(\_,\ x,\ \_,\ \_) \\
\wedge & z = \texttt{MmMapLockedPages}(\_,\ y) \\
\wedge & z[\_] = f \\
\wedge & \mathit{InsideCodeSection}(f)
\end{array}
$$

$$
\begin{array}{ll}
 & y = \text{0x00012018}(\_,\ \text{0x11223344},\ \_,\ \_) \\
\wedge & z = \text{0x00012010}(\_,\ y) \\
\wedge & z[\_] = f \\
\wedge & \mathit{InsideCodeSection}(f)
\end{array}
$$

행동의 표현과 시그니처가 일치하는지 비교

SEOUL NATIONAL UNIVERSITY
Programming Research Laboratory