Virtual Machine Protection Technology and AV industry

Zhenxiang Jim Wang

Microsoft Malware Protection Center

Intro

- Joined Microsoft in 2007
- Main work in Microsoft:
  - Static unpacker development.
    - Finished more than 10 static unpackers, including: Molebox, PECompact, PESpin, SVKP, ASProtect, etc
  - Virtual machine technology analysis/research

- MMPC
  - Microsoft Malware Protection Center (MMPC) established in 2006.
  - Partner with other MS security teams (MSRC, WLSP/SmartScreen, etc.)
  - Responsible for protecting users from malicious threats.
  - Provide core Antimalware technology to Microsoft Security Essentials™、Microsoft® Windows® Defender、Malicious Software Removal Tool, and Forefront™ products.

Pervasive Virtualized Packers Affect AV industry

- Agenda
  - Introduction
  - The Inherent Ability of VM to Defeat Emulation
    - Case Study
  - VM to Dominate Packers
  - The Pervasive Virtualized Packers Defeats Static Unpacking
  - Countermeasure
  - False Positive
  - Bonus Slides – Case Study: Asprotect Stolen Code & Its VM

Introduction

Packer Generations & VM Protection Technology

Introduction: Packers and Generations

- **Compressor** UPX, ASPack

- **Protector** Asprotect, SVK Protector

- **VM Protection system or virtualised packers**

  Themida, VMProtect.

- Need to clarify, ASProtect should be considered as a virtualized packer rather than Protector, because there are 4 VMs used in it.

Introduction:

Characteristics and Usage of VM in Packers

- Virtualization is not new technology
  - Used in different fields to virtualise resource, CPU and application, etc.

- In packers, virtualization is used to defeat reverse engineering.
  - Subverts the concept of traditional packers
  - Original instructions are converted to VM instructions and removed permanently
  - VM instruction are interpreted to execute

- Virtualization techniques in packers can be used to protect:
  - Critical function/code snippet
  - Specific instructions, often used in specific situations.
    For example, in Asprotect, two VMs are used to protect special instructions, such as JCC, JMP, CALL etc, in advanced import protection and stolen code
    - See also: Bonus slide about Asprotect stolen code.

Introduction: VM Implementation

- The following components are necessary to implement a VM
  - **VM API**
    - Used to enter/exit VM. Usually, you cannot expect to find a CALL instruction☺
    - The code to enter/exit VM can be generated at packing time(Themida, VMProtect, ASProtect) or at runtime time (ASProtect)
  - **VM Context**
    - Contains all info to emulate instructions, such as: (1)VM EIP; (2)The buffer to exchange register values between VM and real CPU; (3)VM handlers info; and (4)other specific info.
  - **VM Handler**
    - VM handlers are used to decode and execute VM instructions

Introduction: Obfuscation, the Foundation

- To analyze a VM
  - Understand how VM handlers work and determine the functions of all VM handlers
  - Collect the detailed information about each VM handler
- VM handlers play a critical role in the process of protecting VM from reverse engineering
  - If VM handlers are not safe, the VM is not safe and the applications protected with it will be unsafe
- Obfuscation techniques make the handlers powerful
  - VM handler is usually small and the instructions are straightforward, but obfuscation will make it larger and difficult to understand

Unpacking: Status Quo

- How to deal with packed samples is one of the most challenging problems AV industry faces.
  - Packers protect more than 80% of all existing malware.

- The techniques to deal with packers
  - Generic unpacking
    - Traditional emulator and DT. Hereinafter called ***emulator***
    - Slow
    - Generic
  - Static unpacking
    - Specific
    - Fast
    - Long development time
  - The hybrid approach

VM Defeats Generic Unpacking : The Inherent Ability of VM to Defeat Emulation

VM Defeats Generic Unpacking

- The emulators suffer resource exhaustion when trying to run through virtualized packers.

- Time to emulate a sample packed by a virtualized packer is often too long to tolerate, especially for on-access scan.

Case study: Themida VM Implementation

Case Study: Themida - Patterns

- Patterns are widely used in virtualized / obfuscated packers, including Themida.
- What's a pattern?
  - An Instruction snippet
  - Used repeatedly
  - Makes analysis hard
  - Equivalent to a shorter instruction snippet

Case Study: Themida
Types of Patterns

- **Junk Pattern**
  - Does nothing and can be removed safely
- **Instruction-level pattern**
  - Is equivalent to a single instruction
  - Can be replaced by its equivalent instruction
- **Function-Level Pattern**
  - Equivalent to a shorter instruction snippet

Example: Function-level pattern

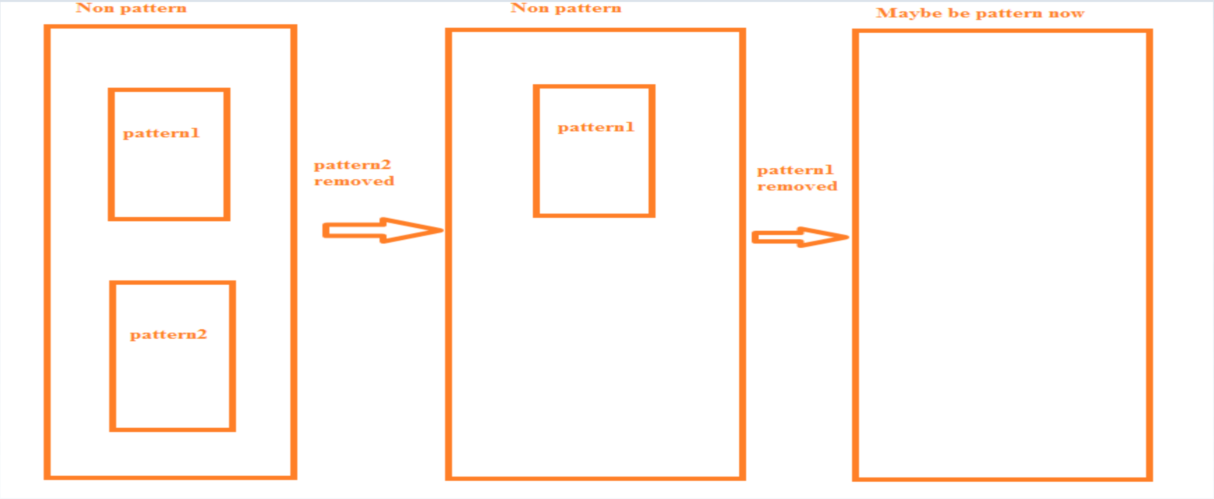
```
Pushf
shr    dword ptr [esp], 6
not    dword ptr [esp]
and    dword ptr [esp], 1
push   eax
push   edx
mov    eax, 12DCB261h
add/sub   eax, 0ED234D44h
mul    dword ptr [esp+8]
lea    eax, [eax+ebp+403767h]
mov    [esp+8], eax
pop    edx
pop    eax
lea    esp, [esp+4]
jmp    dword ptr [esp-4]
```

⇨

```
dec ecx
jnz  xxxx
jmp yyyy
```

Case Study: Themida
Rule to Define Patterns

- **Rule1:** The instruction snippet should be equivalent to a shorter one
- **Rule2:** The instruction snippet should not contain any instruction snippet that can be defined as another pattern. The principle can be named as **MINIMAL** principle.



Case Study: Themida

Apply Patterns at Packing Time

- Applying patterns to obfuscate VM handlers
  - For each instruction to obfuscate in a handler, an equivalent instruction-level pattern is chosen randomly to replace, and then do the same thing for the new code snippet

Example: Apply patterns on the instruction **PUSH EAX**:

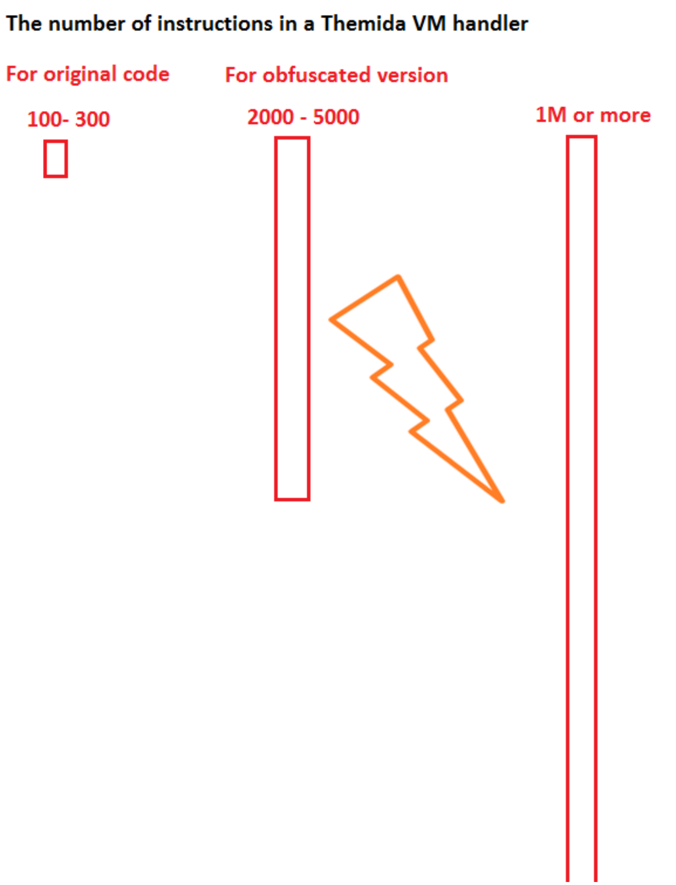
- **Round #1:** Assume choosing the pattern to replace the instruction ***PUSH EAX***
  - **PUSH IMM**
  - **MOV [ESP], REG -> PUSH REG**
  - The instruction will be replaced as
  - **PUSH EAX -> PUSH IMM**
  - **MOV [ESP], EAX**

- **Round #2:** Assume choosing the pattern to replace the instruction ***PUSH IMM***
  - **SUB ESP, 4**
  - **MOV [ESP], IMM -> PUSH IMM**
  - The instruction snippet will be extended to:
  - **PUSH EAX ->PUSH IMM -> SUB ESP, 4**
  - **MOV [ESP], EAX MOV [ESP], IMM**
  - **MOV [ESP], EAX**

- **Round #3:** The instruction **SUB ESP, 4** will be replaced by a randomly chosen pattern, and so on.

Case Study: Themida
The Ability of Anti-Emulation

- Obviously, the implementation mechanism makes it easy to extend the instruction number of a handler to ***1M or more.*** This will defeat generic unpacking easily



VM Tips the Balance

- In the early days, ***signature-based*** approach was used to detect viruses
- Malware authors adopted the ***polymorphic technique*** to counteract the approach.
- ***Emulation technique*** was used to solve the polymorphism issue.
- Malware authors adopt ***virtualization technique*** to defeat emulation.
- Virtualization technique tips the balance of power toward malware authors. ***What is the next story***?

VM to Dominate Packers

VM to Dominate Packers

- Virtualized packers do not occupy a dominant position currently in packer distribution.
- There is an upward trend in the prevalence of virtualized packers in packer distribution.
- Virtualization is becoming a must-have for new developed packers, existing packers are adding the virtualization function.

VM to Dominate Packers

- What if the open-source packer, UPX, the most popular, statistically, adopts VM techniques
- Open-source VM engine
- VM generator
  - Users just need to define syntax of VM instructions.
- It can be predicted reasonably that more and more malware authors will adopt virtualized packers, either existing virtualized packers or custom virtualized packers written by the malware authors themselves, in order to protect their “works” in the near future.

Pervasive VM Defeats Static Unpacking

What About Static Unpacking

- Static unpacking development focus on the packers that
  - Cannot be emulated
  - Takes a long time to emulate
  - Significant performance improvement because of prevalence

- It is still feasible to develop a static unpacker for limited number of prevalent packers, but …

- We may not have enough resources to analyze and optimize numerous unknown virtualized packers even with the help of de-obfuscation tools.

  The prevalence of custom virtualized packers will make static unpacking techniques unfeasible.
  - For example, it took several months to implement Asprotect static unpacker because there are more than 160 versions. Asprotect has a long history. But for custom packers, you will find 160 versions in a shorter period.

Countermeasure

Strategic improvement

Technical improvement

Strategic - Change the Ecosystem

- AV is passive now
- Collaborate with commercial packer vendors
- Get help from the published application vendors
  - If they adopt VM/obfuscation techniques in their applications

Strategic - Commercial Packers

- Blacklist all samples packed with unlicensed commercial packers(Shareware)
- Blacklist licensed packers used in malware
- Blacklist all samples packed with pirated commercial packers.
  - Currently, some AV vendors collect the licensed info of samples to determine if they are packed by a pirated packer ***in their own way***. We need a more ***robust, consistent*** mechanism to identify the pirated packers.

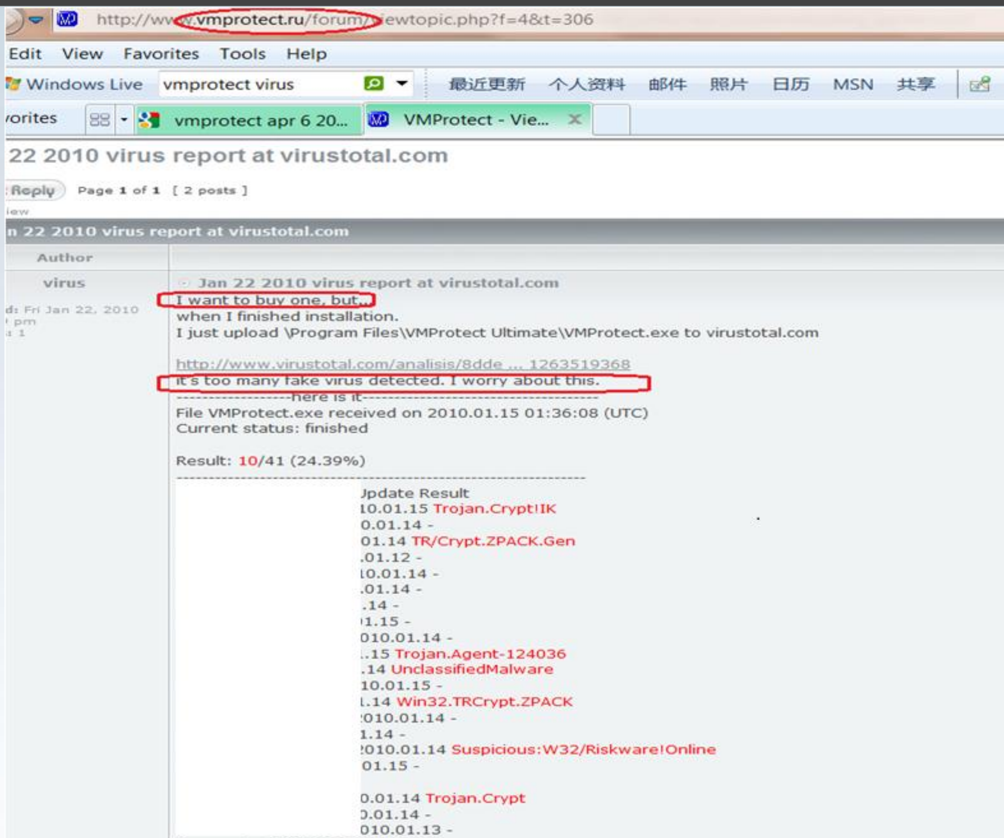
Strategic - Commercial Packers

```
00000000 74 65 63 74 20 4C 69 63+                db 0Dh,0Ah
00000000 65 6E 73 65 20 44 75 6D+                db '[YouProductName], [1.0], ',0Dh,0Ah
00000000 70 0D 0A 0D 0A 5B 59 6F+                db '[1.4 build 01.26 Beta], [Teddy Rogers].',0Dh,0Ah
00000000 75 50 72 6F 64 75 63 74+seg000          ends
00000000 4E 61 6D 65 5D 2C 20 5B+
00000000 31 2E 30 5D 2C 20 0D 0A+
```

Teddy Rogers is the site administrator of www.tuts4you.com

```
65 6E 73 65 20 44 75 6D+                db '[ApplicationName], [Version], ',0Dh,0Ah
70 0D 0A 0D 0A 5B 41 70+                db '[2.3 build 04.26 Beta], [stephenteh',0Dh,0Ah
70 6C 69 63 61 74 69 6F+                db 'TEAM RESURRECTiON]--',0Dh,0Ah
6E 4E 61 6D 65 5D 2C 20+seg000          ends
5B 56 65 72 73 69 6F 6E+
```

Strategic - Commercial Packers



Packer vendors should have motivation to provide more help ☺

Strategic – Handling VM Apps

- Report to White List Association
- Digitally sign their applications

Technical – Invest in Unpacking

- Most prevalent virtualized commercial packer
- It is worth investing in
  - Developing static unpacker
    - Asprotect static unpacking: including restoring virtualized x86 instructions, recovering stolen OEP, stolen functions, missing Delphi init/term table etc, the unpacked file can run normally
      - See also: Bonus slide: Case Study: Asprotect stolen code & its VM
  - The hybrid approach of generic unpacking and static unpacking. Implement VM statically on the basis of emulation.
    - Themida: recover virtualized x86 instructions
- Numerous unknown virtualized custom packers.
  - Generic unpacking, static unpacking and the hybrid will fail.

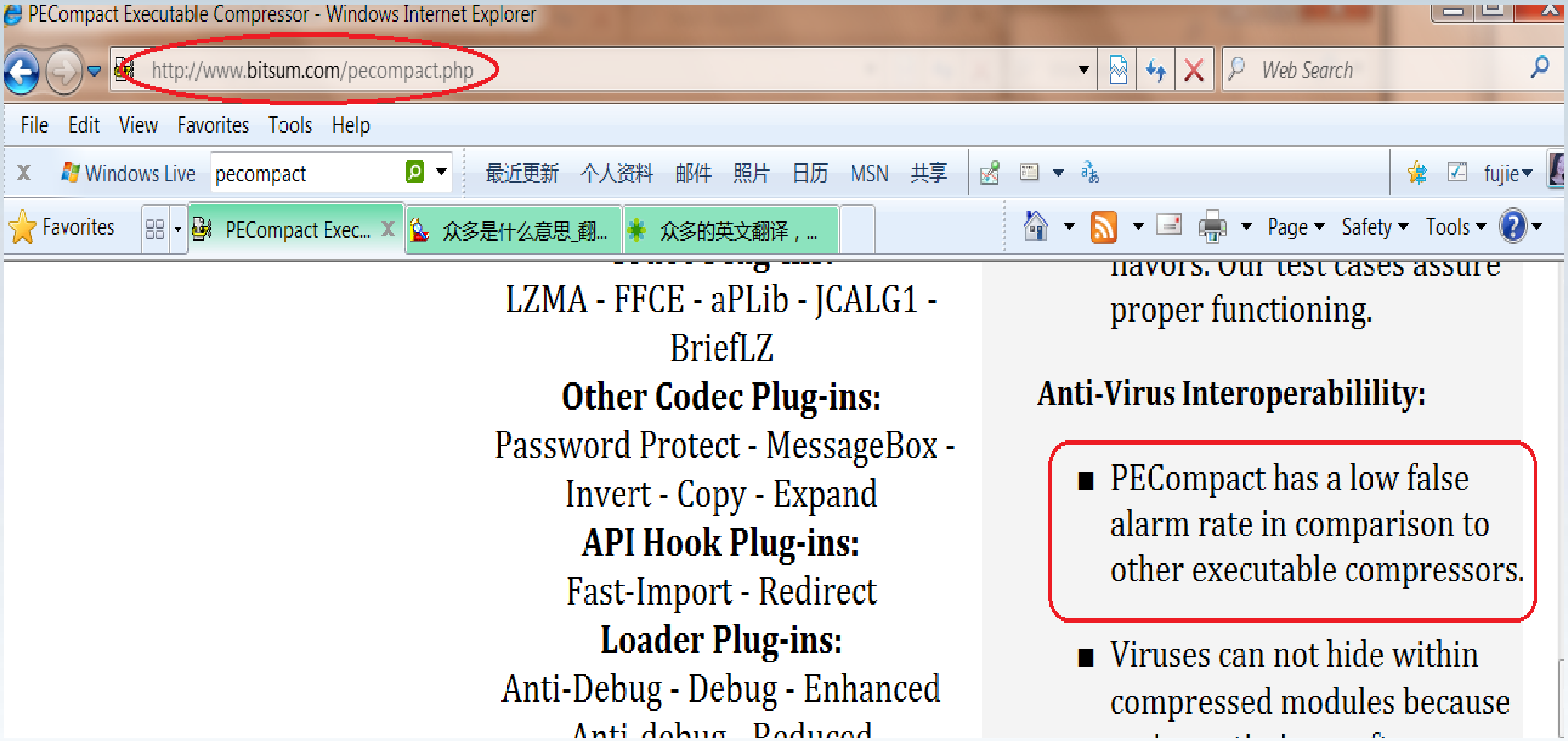
Technical – Deal with Unknown

- If emulator cannot run through, maybe we can adopt the combination of full-fledged emulation technique and behavior analysis.

  - Full-fledged emulator will defeat the anti-emulation and virtualized code
  - APIs will just use to record behaviors.
  - This should be an additional component.

False Positive

An interesting Note on PECompact



It implies at least two things:

- ❖There are a few false positives
- ❖There are false positives, even for compressors

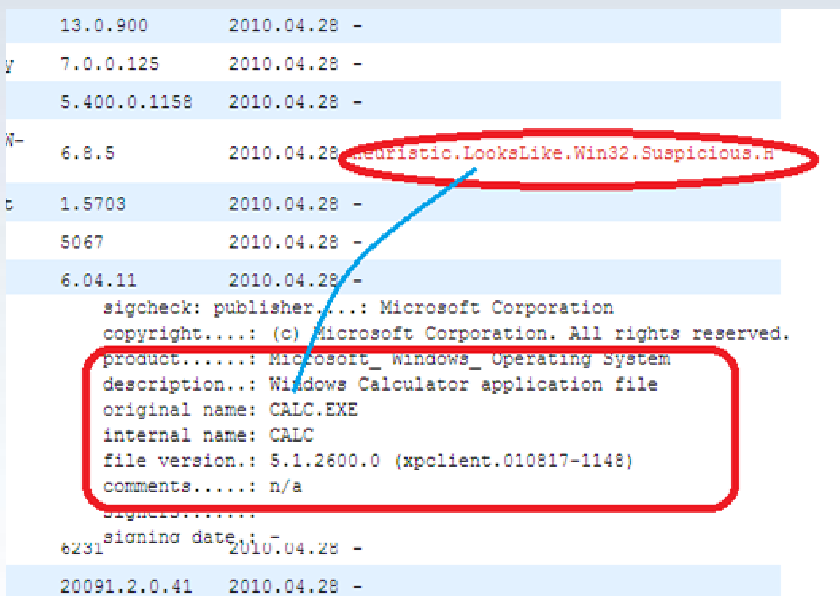
False Positive

- Even now, we can find many false positives.
  - These false positives may be due to packer blacklisting. Some in the industry may argue that the benefits for protection overweigh the harm caused by FPs. Users may disagree.

- Industry likely continue to see false positives of this sort in the future.

False Positive

- It will be much more difficult to avoid false positive completely when adopting behavior analysis techniques



	13.0.900	2010.04.28	-
y	7.0.0.125	2010.04.28	-
	5.400.0.1158	2010.04.28	-
N-	6.8.5	2010.04.28	Heuristic.LooksLike.Win32.Suspicious.H
t	1.5703	2010.04.28	-
	5067	2010.04.28	-
	6.04.11	2010.04.28	-

```
sigcheck: publisher....: Microsoft Corporation
copyright....: (c) Microsoft Corporation. All rights reserved.
product......: Microsoft_ Windows_ Operating System
description..: Windows Calculator application file
original name: CALC.EXE
internal name: CALC
file version.: 5.1.2600.0 (xpclient.010817-1148)
comments.....: n/a
signers......:
signing date.: -
```

	6231	2010.04.28	-
	20091.2.0.41	2010.04.28	-

A Word on Cloud Computing

- Many Web-based application/platform available
- Security issues continue to concern people, because they will lose control of their information in the cloud computing environment.
- But cloud computing might be a way to defeat rampant virtualized viruses on the desktop.

Conclusion

- With the prevalence of virtual machine protection techniques, AV industry might be at a turning point
- We may need to take a more active strategy
- We need new techniques to deal with virtualized packers, just like adopting emulation technique to deal with polymorphic viruses.

Thank You

jimwan@microsoft.com

Bonus Slides:
Case study: ASProtect stolen code & its VM

Virtual Machines in ASProtect

- There are four VMs in Asprotect.
  - Two of them are used to protect critical functions
  - One is used to protect stolen code
  - One is used to protect advanced import protection(AIP)

- Two completely different implementations
  - Soft CPU to protect critical functions
  - Standard VM to protect ***stolen code*** & Advanced import protection(AIP)

What’s Stolen code

- The original code snippet is placed somewhere else in the file or a dynamically allocated memory

- A JMP instruction to the stolen code is inserted at the beginning of the original code snippet

- The stolen code is often protected using obfuscation technology

- **Stolen OEP(Original entry point) is a special case**
  - The address of stolen OEP is often computed dynamically

Asprotect Steals Many Code in Different Ways

- ***Missing functions***. Some functions are replaced by equivalent obfuscated code snippets
- ***The function to process the init table in Delphi applications*** is replaced by an obfuscated code snippet and the init table is destroyed.
- ***The OEP and the licensed functions*** are stolen in a much more complicated way.

How ASProtect Steals OEP code

- Six steps:
  - Scan the OEP code and generate new basic blocks for CALL, JMP & JCC instructions
  - Obfuscate the OEP code snippet
    - Use many different de-optimization techniques, such as def-use chain, const expand, junk patterns, etc.
  - Divides the obfuscated code snippet into different block randomly
  - Virtualize some special instructions, such as JCC/JMP, CMP, etc
  - Encrypt the return address of the CALL instructions inside the code snippet
  - Encrypt the obfuscated stolen OEP code

How to Recover Stolen OEP

- The reverse process to recovering the equivalent OEP code snippet is as follows:
  - Decrypt the obfuscated code snippet
  - Recover virtual machine emulated instructions, including CALL instructions
  - Generate correct return address for the emulated CALL instructions
  - De-obfuscate the code snippet
    - Scan the code snippet and generate the intermediate representation for each instruction
    - De-obfuscate based on the IR format instructions
    - Generate opcode for de-obfuscated instructions, in IR format
  - Compute target addresses of CALL/JCC/JMP instructions
  - Generate opcodes for all de-obfuscated instructions

Microsoft® Malware Protection Center
Threat Research and Response

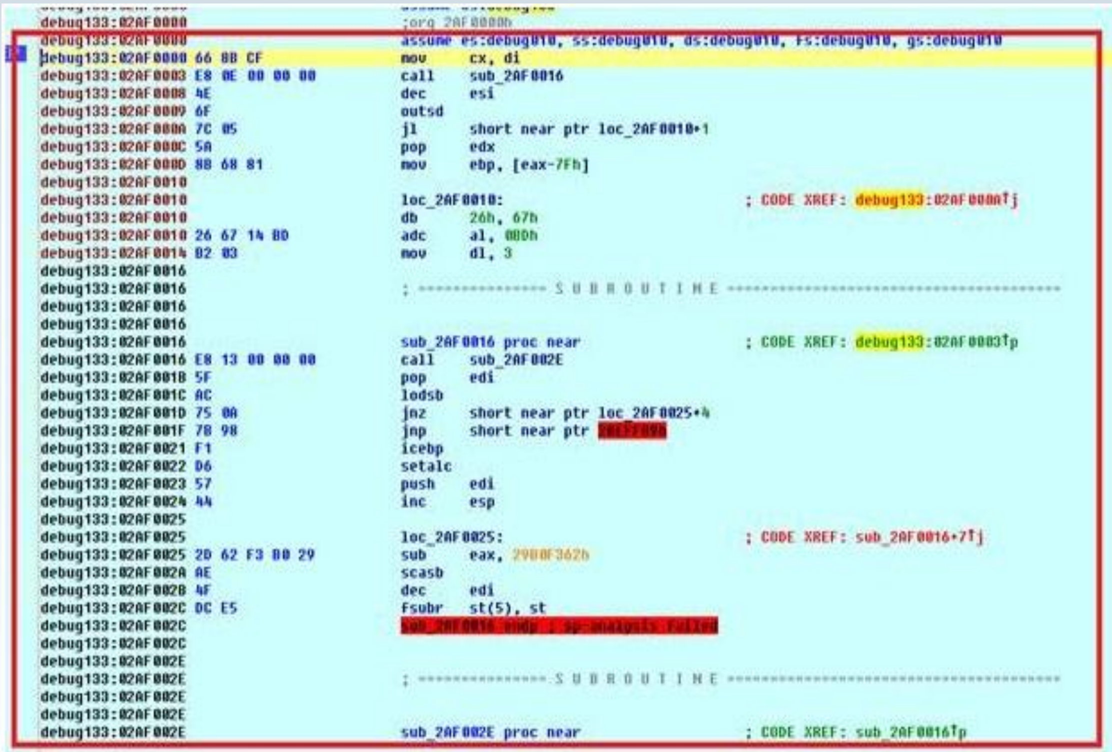
An Example

The original entry point of ATTRIB.EXE in XP

```
text:010022A8
text:010022A8 6A 28                     push    28h
text:010022AA 68 68 12 00 01            push    offset stru_1001268
text:010022AF E8 90 01 00 00            call    __SEH_prolog
text:010022B4 33 FF                     xor     edi, edi
text:010022B6 57                        push    edi             ; lpModuleName
text:010022B7 FF 15 08 10 00 01         call    ds:GetModuleHandleA
text:010022BD 66 81 38 4D 5A            cmp     word ptr [eax], 5A4Dh
text:010022C2 75 1F                     jnz     short loc_10022E3
text:010022C4 8B 48 3C                  mov     ecx, [eax+3Ch]
text:010022C7 03 C8                     add     ecx, eax
text:010022C9 81 39 50 45 00 00         cmp     dword ptr [ecx], 4550h
text:010022CF 75 12                     jnz     short loc_10022E3
text:010022D1 0F B7 41 18               movzx   eax, word ptr [ecx+18h]
text:010022D5 3D 0B 01 00 00            cmp     eax, 10Bh
text:010022DA 74 1F                     jz      short loc_10022FB
text:010022DC 3D 0B 02 00 00            cmp     eax, 20Bh
text:010022E1 74 05                     jz      short loc_10022E8
text:010022E3
text:010022E3           loc_10022E3:                    ; CODE XREF: start+1A↑j
text:010022E3                                           ; start+27↑j ...
text:010022E3 89 7D E4                  mov     [ebp+var_1C], edi
text:010022E6 EB 27                     jmp     short loc_100230F
text:010022E8           ; ---------------------------------------------------------
text:010022E8
text:010022E8           loc_10022E8:                    ; CODE XREF: start+39↑j
text:010022E8 83 B9 84 00 00 00 0E      cmp     dword ptr [ecx+84h], 0Eh
text:010022EF 76 F2                     jbe     short loc_10022E3
text:010022F1 33 C0                     xor     eax, eax
text:010022F3 39 B9 F8 00 00 00         cmp     [ecx+0F8h], edi
text:010022F9 EB 0E                     jmp     short loc_1002309
text:010022FB           ; ---------------------------------------------------------
text:010022FB
text:010022FB           loc_10022FB:                    ; CODE XREF: start+32↑j
text:010022FB 83 79 74 0E               cmp     dword ptr [ecx+74h], 0Eh
text:010022FF 76 E2                     jbe     short loc_10022E3
text:01002301 33 C0                     xor     eax, eax
text:01002303 39 B9 E8 00 00 00         cmp     [ecx+0E8h], edi
text:01002309
text:01002309           loc_1002309:                    ; CODE XREF: start+51↑j
text:01002309 0F 95 C0                  setnz   al
text:0100230C 89 45 E4                  mov     [ebp+var_1C], eax
text:0100230F
```

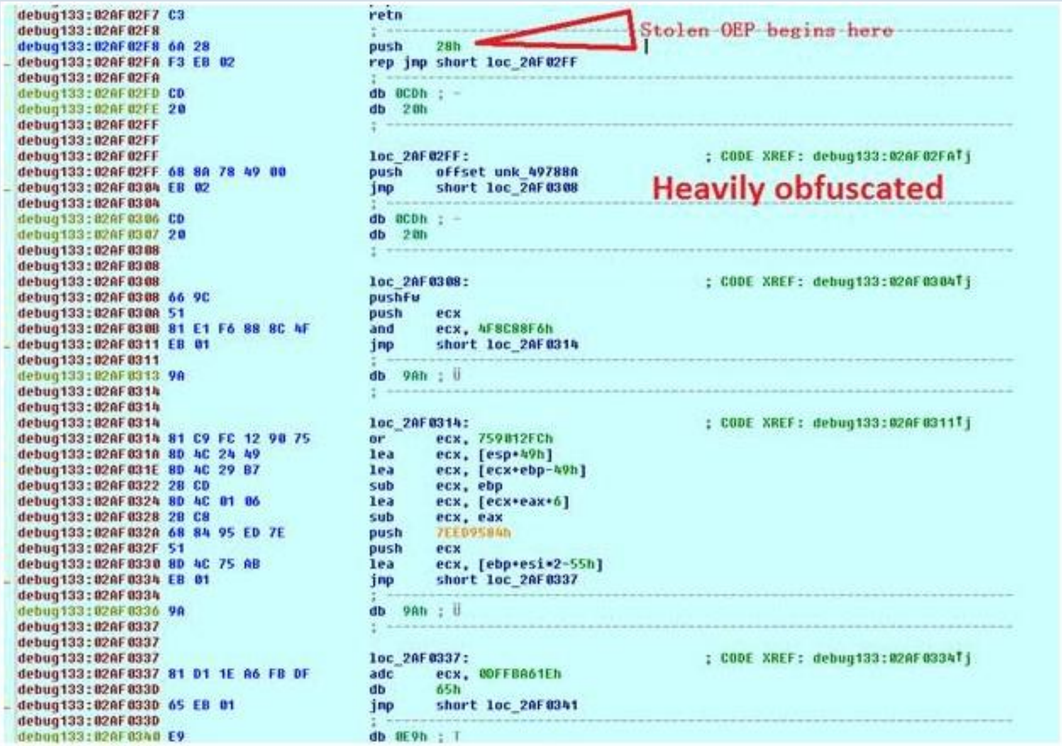
Decrypt Routine

- The routine to decrypt the stolen OEP



```
debug133:02AF0000                         ;org 2AF0000h
debug133:02AF0000                         assume es:debug010, ss:debug010, ds:debug010, fs:debug010, gs:debug010
debug133:02AF0000 66 8B CF                mov     cx, di
debug133:02AF0003 E8 0E 00 00 00          call    sub_2AF0016
debug133:02AF0008 4E                      dec     esi
debug133:02AF0009 6F                      outsd
debug133:02AF000A 7C 05                   jl      short near ptr loc_2AF0010+1
debug133:02AF000C 5A                      pop     edx
debug133:02AF000D 8B 68 81                mov     ebp, [eax-7Fh]
debug133:02AF0010
debug133:02AF0010                         loc_2AF0010:                  ; CODE XREF: debug133:02AF000A↑j
debug133:02AF0010                         db      26h, 67h
debug133:02AF0010 26 67 14 BD             adc     al, 0BDh
debug133:02AF0014 B2 03                   mov     dl, 3
debug133:02AF0016
debug133:02AF0016                         ; =============== S U B R O U T I N E =======================================
debug133:02AF0016
debug133:02AF0016
debug133:02AF0016                         sub_2AF0016 proc near         ; CODE XREF: debug133:02AF0003↑p
debug133:02AF0016 E8 13 00 00 00          call    sub_2AF002E
debug133:02AF001B 5F                      pop     edi
debug133:02AF001C AC                      lodsb
debug133:02AF001D 75 0A                   jnz     short near ptr loc_2AF0025+4
debug133:02AF001F 7B 98                   jnp     short near ptr [illegible]
debug133:02AF0021 F1                      icebp
debug133:02AF0022 D6                      setalc
debug133:02AF0023 57                      push    edi
debug133:02AF0024 44                      inc     esp
debug133:02AF0025
debug133:02AF0025                         loc_2AF0025:                  ; CODE XREF: sub_2AF0016+7↑j
debug133:02AF0025 2D 62 F3 B0 29          sub     eax, 29B0F362h
debug133:02AF002A AE                      scasb
debug133:02AF002B 4F                      dec     edi
debug133:02AF002C DC E5                   fsubr   st(5), st
debug133:02AF002C                         sub_2AF0016 endp ; sp-analysis failed
debug133:02AF002C
debug133:02AF002E
debug133:02AF002E                         ; =============== S U B R O U T I N E =======================================
debug133:02AF002E
debug133:02AF002E
debug133:02AF002E                         sub_2AF002E proc near         ; CODE XREF: sub_2AF0016↑p
```

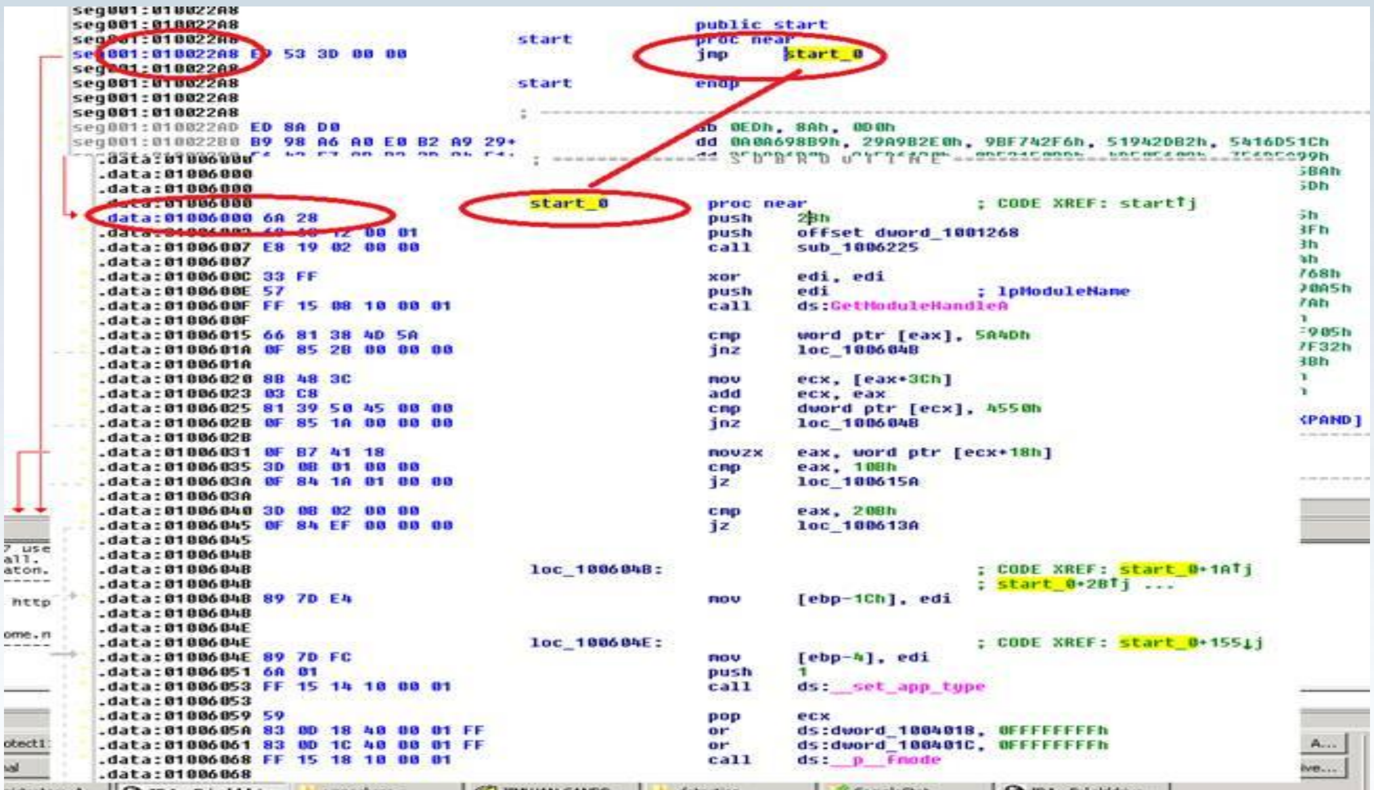
Decrypted Stolen OEP



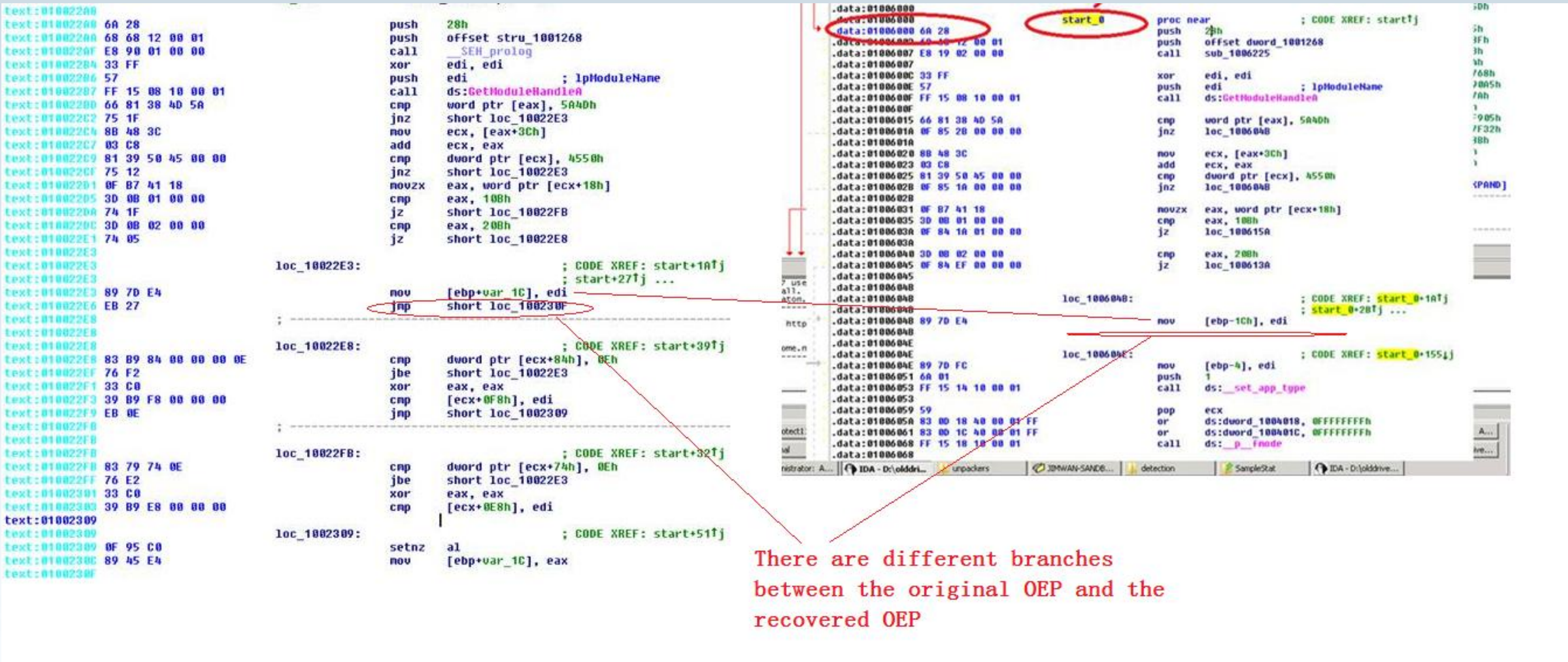
Emulate Special Instructions

The virtual machine technique is used to emulate some special instructions

Recovered OEP code



Comparison


There are different branches
between the original OEP and the
recovered OEP

Microsoft®

Your potential. Our passion.™