Virtual Machine Protection Technology and AV industry

Zhenxiang Jim Wang

Microsoft Malware Protection Center
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.
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.
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.
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

Introduction:
Obfuscation, the Foundation
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
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
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
```
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.
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.
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

- Introduction
- The Inherent Ability of VM to Defeat Emulation
  - Case Study
- VM to Dominate Packers
- Pervasive VM defeats Static Unpacking
- Countermeasure
- False Positive
- Bonus Slides – Case Study: Asprotect stolen code & its VM
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.
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

The Inherent ability to defeat emulation

Case Study: Asprotect stolen code & its VM

Bonus Slides - Case Study: Asprotect stolen code & its VM

False Positive

Countermeasure

VM To Dominate Packers

Case Study

The inherent ability to defeat emulation
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
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.
Teddy Rogers is the site administrator of www.tuts4you.com
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
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.
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

Anti-Virus Interoperability:

- PECompact has a low false alarm rate in comparison to other executable compressors.
- Viruses can not hide within compressed modules because they are not invisible.
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 outweigh the harm caused by FPs. Users may disagree.

Industry likely continue to see false positives of this sort in the future.
It will be much more difficult to avoid false positive completely when adopting behavior analysis techniques.
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**

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Inherent Ability of VM to Defeat Emulation</td>
</tr>
<tr>
<td>Case Study</td>
</tr>
<tr>
<td>VM To Dominate Packers</td>
</tr>
<tr>
<td>Pervasive VM defeats Static Unpacking</td>
</tr>
<tr>
<td>Countermeasure</td>
</tr>
<tr>
<td>False Positive</td>
</tr>
<tr>
<td><strong>Bonus Slides – Case Study: Asprotect stolen code &amp; its VM</strong></td>
</tr>
</tbody>
</table>
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.
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
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
An Example

The original entry point of ATTRIB.EXE in XP
Decrypt Routine

- The routine to decrypt the stolen OEP
Decrypted Stolen OEP

Heavily obfuscated
The virtual machine technique is used to emulate some special instructions.
Comparison

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