A Framework for Understanding Dynamic Anti-Analysis Defenses

Jing Qiu  Babak Yadegari  Brian Johannesmeyer
Saumya Debray  Xiaohong Su

Harbin Institute of Technology, Harbin, China
The University of Arizona, Tucson, US

Dec. 9, 2014
Malware combine a variety of defenses to avoid detection and hinder analysis.
We propose a general information-flow-based framework to detect anti-analysis defenses.
Self-Checksumming

Computing a checksum

- Simple comparison
- Checksum-based code unpacking

Observation → Validation → Response

Normal Execution

Abnormal Execution
Timing-Based Anti-Emulation

Observation → Information Flow → Validation → Information Flow → Response

$t_0 = \text{clock}()$
...
$t_1 = \text{clock}()$
...

if $(t_1 - t_0 > \text{threshold})$
if $(t_1 - t_0 > \text{threshold})$
{
// response
}
// normal execution

Normal Execution
Abnormal Execution
Anti-Debugging using ProcessDebugPort

NtQueryInformationProcess(-1, ProcessDebugPort, &retVal, ...);

if (retVal != 0)
{
  // Debugger Detected
} else {
  // Normal execution
}
Anti-VM - Detect VMWare

Observation

Validation

Response

Information Flow

Information Flow

Normal Execution

Abnormal Execution

mov eax, `VMXh`
mov ecx, 0ah
mov dx, `VX`
in eax, dx

cmp ebx, `VMXh`
j e detected
Stages of the framework can be used in different combinations to create and understand new defenses.

- Code checksum
- Time clock
- NtQueryInformation
- Process
- Exception generation
- ...

- Comparison instruction
- Flags register
- manipulation
- Indirect jump
- Other tricky math...

- Affect control flow
- Use as unpacking key
- Correctness of output
- ...

Observation  

Validation  

Response
Detection: Self-Checksumming (1/6)
Detection: Self-Checksumming (2/6)
Detection: Self-Checksumming (3/6)

- C
- C'
- Unpacking
- Checksum
- Validation
Detection: Self-Checksumming (4/6)
Detection: Timing-Based Anti-Emulation

```
Trace
...
\[ t_0 = \text{clock()} \]
...
\[ t_1 = \text{clock()} \]
...
verifier(t_0, t_1)
```

Forward Taint
Detection: Anti-Debugging using ProcessDebugPort

Trace

...mov eax, offset RetVal
**push eax**
push 7 ; ProcessDebugPort
...
call NtQueryInformationProcess
...

**cmp RetVal, 0**
jne DebuggerDetected

Forward Taint
Detection: Anti-VM - VMware

Trace

...  
mov eax, `VMXh`
mov ecx, 0ah
mov dx, `VX`
in eax, dx
...  

cmp ebx, `VMXh`
je detected
Forward Taint
50-guards: 50 different interleaving self-checksumming guards
decrypt-key: a checksum as a code decryption key
chksum-md5: a checksum to generate an MD5 initialization constant
time-md5: time-based defense
timing-themida & timing-obsidium packed with Themida and Obsidium
### Table 1: Evaluation Result

<table>
<thead>
<tr>
<th>Program</th>
<th>No. of Guards</th>
<th>Analysis Time (sec)</th>
<th>Instructions in Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Found</td>
<td>Ground Truth</td>
<td></td>
</tr>
<tr>
<td>50-guards</td>
<td>50</td>
<td>50</td>
<td>95</td>
</tr>
<tr>
<td>decrypt-key</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>chksum-md5</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>time-md5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>timing-obsidium</td>
<td>0</td>
<td>Unknown</td>
<td>347</td>
</tr>
<tr>
<td>timing-themida</td>
<td>2</td>
<td>Unknown</td>
<td>223</td>
</tr>
</tbody>
</table>
Limitation

- **Low code coverage**
  - The code coverage could be improved by multiple path exploration
  - or generating inputs by symbolic execution

- **Large scale trace files**
  - Recording relevant instructions
  - or parallel processing
Challenges - Implicit Flow

Example

```plaintext
x=0; z=0;
if(y=0) then x=1; else z=1; endif
if(x=0) then w=0; endif
if(z=0) then w=1; endif
```
Challenges - False Positives

Self-Checksumming: Data compression
Challenges - False Positives

Timing-based anti-emulation: Benchmark programs

Example

```c
// Example code

double t = clock();
...

t = clock() - t
if (t < 10) score = 100;
else if (t < 20) score = 90;
...
```
Conclusion

- We describe an information-flow-based framework for understanding a wide variety of anti-analysis defenses.
- Self-checksumming, timing-based anti-emulation, anti-debugging and anti-VM defenses are instances of this framework.
- Experimental results shows that this framework is effective.
Related Work

- In 2005, Wurster et al. employs hardware assisted techniques to bypass the self-checksumming defense.
- In 2005, Giffin et al. show that self-modifying code can be used to detect the attack Wurster et al. proposed.
- In 2008, Brumley et al. use a combination of dynamic binary instrumentation and mixed symbolic and concrete execution, to identify behavior that is dependent on environmental triggers.
- In 2006, Crandall et al. use a combination of VM-based timer perturbation and symbolic execution to discover time bombs in malware.
- In 2010, Lindorfer et al. and Balzarotti et al. discuss detecting environment-dependent behavior in native malware by comparing multiple executions in different environments.
Questions?
checksum = compute_checksum(CODE);
for (i = 0; i < size; i++)
    CODE[i] -= checksum;
```c
int cksum = compute_checksum(CODE);
...
// This value should be 0x67452301;
mdContext->buf[0] = cksum + 0x6740E9CB;
...
printf("%s", md5_str);
...
DWORD t1 = GetTickCount();
MD5();

if (GetTickCount() - t1 > 50)
{
    printf("I am traced.\n");
}