Analysis of Packed and Obfuscated Program Binaries

Kevin Roundy
University of Wisconsin

Paradyn/Dyninst Week
April 27-28, 2009
Analysis-Resistant Malware

GhostNet
  - Hacked embassies all over SE Asia

Conficker

90% of malware resists analysis \[1\]
  - Dynamically generated code
  - Overwritten code
  - Obfuscated control flow

\[1\] Packer (r)evolution, *Panda Research, 2008*
Dynamically Generated Code

75% of malware apply tools to pack their binaries, generating code at runtime [1]

Original Malware Binary

Packed Binary

Entry Point

Obfuscated bootstrap code

Original binary initially compressed or encrypted

Payload program is generated at runtime

Control transfer to unpacked code
Dynamically Generated Code

75% of malware apply tools to pack their binaries, generating code at runtime [1]

Original Malware Binary

Packer growth rate of 6-8% per month [2]

75% of new malware uses custom packing [1]

Packed Binary

payload program is generated at runtime

custom transfer to unpacked code

Overwritten Code

Self-modifying programs overwrite code at runtime

CFG - initial

A
  jmp B

B
  mov C,[A+1]
  jmp A

CFG - after write

A
  jmp C

B
  mov C,[A+1]
  jmp A

invalidated code

new code

unreachable code
Obfuscated Control Flow

Statically un-analyzable control flow hides code

```
jmp eax  →  call ptr[eax]  →  push 401000  →  div eax, 0
          ↓               ↓               ↓               ↓
          ?               ?               ret             div0 handler
```

©2009 Kevin Roundy
Analysis of Packed and Obfuscated Program Binaries
Our Goal

Analyze the code, monitor and control the program

Solution: Hybrid analysis

- Parsing
- Instrumentation-based code discovery
- Code overwrite detection
- Signal- and exception-monitoring

static techniques
dynamic techniques
Control Transfers to Instrument

Invalid control transfers
- call 401000
  → Invalid Region

Indirect jumps/calls
- jmp eax
  →?
- call ptr [eax]
  →?

Return instructions
- push eax
  → ret
Instrumentation-Based Discovery

Dyninst

stopThreadExpr

- Evaluates instrumentation predicate
  - `constExpr`
  - `dynamicTargetExpr`

- Causes Dyninst to invoke synchronous callback
  - `cb(instrumentedAddr, targetAddr)`
Code Discovery Algorithm

**Hybrid code discovery:**

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
**Code Discovery Algorithm**

*Hybrid code discovery:*

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
Code Discovery Algorithm

*Hybrid code discovery:*

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
Code Discovery Algorithm

Hybrid code discovery:

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
**Code Discovery Algorithm**

*Hybrid code discovery:*

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
**Code Discovery Algorithm**

Hybrid code discovery:

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
Code Discovery Algorithm

Hybrid code discovery:

- Parse from known entry points
- Instrument control flow that may lead to new code
- Resume execution
- Instrumentation discovers new entry points
Code Discovery Algorithm

Hybrid code discovery:

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
Code Discovery Algorithm

Hybrid code discovery:

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
**Code Discovery Algorithm**

*Hybrid code discovery:*

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution
4. Instrumentation discovers new entry points
Overwritten Code Discovery

Dyninst

code write handler

Overwrite Detection

Possible strategies

- Check each instruction in the trace for changes [1]
- Monitor writes to code regions

Page-level detection [2]

- Remove write permissions from code pages
- Write to code causes exception
- Handle exception

[1] Royal et al. PolyUnpack. ACSAC '06
Overwritten Code Discovery

Dyninst

code write handler

CFG update routine

Updating the CFG

- emulate write instruction
- remove overwritten and unreachable blocks
- parse at entry points to overwritten regions
Overwritten Code Discovery

Dyninst

- code write handler
- CFG update routine

Updating the CFG

- emulate write instruction
- remove overwritten and unreachable blocks
- parse at entry points to overwritten regions
Overwritten Code Discovery

Dyninst

- code write handler
- CFG update routine

Optimizations

Cases to consider

- large incremental overwrites
- writes to data
Overwritten Code Discovery

Dyninst

Optimizations

Cases to consider

- large incremental overwrites
- writes to data

Delaying the update

- until write routine terminates
Overwritten Code Discovery

Dyninst

- code write handler
- CFG update routine

Delayed updates

Two components
1. Handle overwrite signal
2. Update CFG when writes end
Overwritten Code Discovery

Dyninst

Delayed updates

Two components

1. Handle overwrite signal
   a) instrument write loop exits
   b) copy overwritten page
   c) restore write permissions
   d) resume execution

2. Update CFG when writes end
Overwritten Code Discovery

Dyninst

code write handler

CFG update routine

Delayed updates

Two components

1. Handle overwrite signal
   a) instrument write loop exits
   b) copy overwritten page
   c) restore write permissions
   d) resume execution

2. Update CFG when writes end
   a) remove overwritten and unreachable blocks
   b) parse at entry points to overwritten regions
   c) remove write permissions
Overwritten Code Discovery

Dyninst

code write handler

CFG update routine

Delayed updates

Two components

1. Handle overwrite signal
   a) instrument write loop exits
   b) copy overwritten page
   c) restore write permissions
   d) resume execution

2. Update CFG when writes end
   a) remove overwritten and unreachable blocks
   b) parse at entry points to overwritten regions
   c) remove write permissions
Signal and Exception Handler Discovery

Monitored Program

Div0 signal handler

Dyninst

Operating System

Exception State

load eax, 0
div ebx, eax
add ebx, ebx
...

eip 401002
eax 0
...

©2009 Kevin Roundy  Analysis of Packed and Obfuscated Program Binaries 29
Handler-Based Redirections [1]

Monitored Program

Div0 signal handler

load eax, 0
div ebx, eax
add ebx, ebx
...

Exception State

eip 402d8a
eax 0
...

Dyninst

Operating System

Instrumenting the Handlers

Monitored Program

Dyninst

Operating System

- load eax, 0
- div ebx, eax
- add ebx, ebx

Exception State
- eip 402d8a
- eax 0
- ...

Handler-Based Redirections [1]

Monitored Program

Dyninst

Operating System

Div0 signal handler

Exception State

[eip 402d8a

eax 0

...]


©2009 Kevin Roundy

Analysis of Packed and Obfuscated Program Binaries
Overall Algorithm

1. Parse from known entry points
2. Instrument control flow that may lead to new code
3. Resume execution

- stopThreadExpr (ptr[eax])
- call ptr[eax]
- overwrite
- exception
- div eax, 0
<table>
<thead>
<tr>
<th>Packer</th>
<th>malware market share$^{[1]}$</th>
<th>Dyninst</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPX</td>
<td>9.45%</td>
<td>yes</td>
</tr>
<tr>
<td>PolyEnE</td>
<td>6.21%</td>
<td>yes</td>
</tr>
<tr>
<td>EXEcryptor</td>
<td>4.06%</td>
<td></td>
</tr>
<tr>
<td>Themida</td>
<td>2.95%</td>
<td></td>
</tr>
<tr>
<td>PECompact</td>
<td>2.59%</td>
<td></td>
</tr>
<tr>
<td>Upack</td>
<td>2.08%</td>
<td>yes</td>
</tr>
<tr>
<td>nPack</td>
<td>1.74%</td>
<td>yes</td>
</tr>
<tr>
<td>Aspack</td>
<td>1.29%</td>
<td>yes</td>
</tr>
<tr>
<td>FSG</td>
<td>1.26%</td>
<td>yes</td>
</tr>
<tr>
<td>Asprotect</td>
<td>0.43%</td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td>0.37%</td>
<td></td>
</tr>
<tr>
<td>Yoda's Protector</td>
<td>0.33%</td>
<td></td>
</tr>
<tr>
<td>WinUPack</td>
<td>0.17%</td>
<td>yes</td>
</tr>
<tr>
<td>MEW</td>
<td>0.13%</td>
<td>yes</td>
</tr>
</tbody>
</table>

Tamper-resistance techniques (e.g., self-checksumming)