Generic Unpacking

How to handle modified or unknown PE Compression Engines

Tobias Graf
tobias.graf@ewido.net
ewido networks GmbH & Co. KG

ewido networks
www.ewido.net

Who we are

- Product: ewido security suite
  - Protection against Trojans, Adware, Spyware, ...
- First release: Christmas 2003
- Emulation research since 2002
  - Used for generic unpacking

Agenda

- Motivation
- Generic unpacking
- Typical problems
- Results

Motivation

- How PE compression engines work
- Static unpacking
- When static unpacking engines fail

ewido networks
www.ewido.net

How PE compression engines work

- Malware is detected by searching for known patterns
- Malware is detected by heuristics analysis
- PE compression engines compress or encrypt the content of portable executables
- Compression/encryption cause pattern matching and heuristics analysis fail

How PE compression engines work

- Structure of PE compression engines is usually the same
- Look at a very simple one to show the most important issues
- Lamecrypt encrypts the body by XOR
- Loader is appended to decrypt at runtime

How PE compression engines work

Loader of Lamecrypt

```
mob ebx, 0001D600

decryptloop:
xor [ebx+00401000], 90
dec ebx
cmp ebx, -1
jnz decryptloop

jmp 0041DA48
```

Static unpacking

- Analyze decompress-/decrypt-algorithm
- Decompress/decrypt with specific routine
  - Which PE compression engine was employed?
  - Pick required parameters
  - Invoke the specific routine

Static unpacking

Special routine for Lamecrypt

```
void decrypt_lamecrypt(BYTE* image, DWORD size)
{
   if (checksig(entrypoint+0x0F, “4B83FBFF75F39D”))
   {
      DWORD offset = getDWORD(entryoint+0x0A);
      DWORD size = getDWORD(entrypoint+0x04);
      for (DWORD i=0; i<size; i++)
         image[i] = image[i] ^ 0x90;
   }
}
```

When static unpacking engines fail

- Unknown PE compression engine
  - No specific routine
  - No way to decrypt/decompress
- Modified PE compression engine
  - Detection fails and required parameters cannot be extracted

When static unpacking engines fail

Bagle

- Uses its own PE compression engine to hide the malicious activity
- Could be detected easily by heuristics
  - Even simple compression engines are effective

When static unpacking engines fail

Bagle

```
...
mov esi, 00401000
mov edi, esi
mov ecx, 00004F4B
cld

decryptloop:
lodsb
xor al, 13
stosb
loop decryptloop
```

When static unpacking engines fail

Yodacrypter

- Modified PE compression engine
- Modification is very easy (open source)
- Often used with backdoors

When static unpacking engines fail

Yodacrypter

original loader	modified loader

`pushad`
`call +00`
`pop ebp`
`sub ebp, 00401DF3`
`mov ecx, 97B`
`lea edi, [ebp+00401E3B]`
	`pushad`
	`call +00`
	`pop ebp`
	`sub ebp, 00401F0`
	`mov ecx, 15`
	`add ebp, 3`
	`mov ecx, 97B`
	`lea edi, [ebp+00401E3B]`

When static unpacking engines fail

- Unknown PE compression engines
- Attacks on detection
- Attacks on parameters

Generic unpacking

ewido networks
www.ewido.net

- What do we need?
- How to unpack every PE compression engine
- CPU simulation
- Memory simulation
- Operating system simulation

What do we need?

- Decompress/decrypt PE compression engine without knowing which PE compression engine was employed
- Decompress/decrypt without parameters or algorithm

Unpack every PE compression engine

- Execute the malware, wait a little and perform a memory scan
- Every PE compression engine needs to unpack/decrypt its content in memory
- Ultimate generic unpacker

Unpack every PE compression engine

- Very unsafe!
- Extension: instead of execution
  - Emulation
- If the emulation can achieve a near-perfect simulation of the reality
  - Ultimate and safe generic unpacker

CPU Simulation

- Known to handle polymorphic viruses
- Step by step each instruction is simulated

CPU Simulation


```
mob ebx, 0001D600

decryptloop:

xor [ebx+00401000], 90

dec ebx

cmp ebx, -1

jnz decryptloop
```

set sim_ebx to 0001D600

calculate address, map, xor with 0x90

decrease sim_ebx by one

CPU Simulation

- PE compression engines execute millions of instructions
  - All of them have to be emulated!
- Speed is the most important problem
- 450kb executable packed with UPX
  - 6 million instructions to unpack

Memory Simulation

- Has to be very fast
- Decompression routines transfer lots of bytes
- 450kb packed executable
- Theoretically 900.000 memory operations
- The number of operations often doubles

Memory Simulation

- Need to simulate stack, executable image, heap
  - Needs to be flexible
- Some anti-debugging techniques rely on write access exceptions
- Simulate read/write access

Operating System Simulation

- Import table needs to be rewritten
  - LoadLibrary/GetProcAddress
- Some PE compression engines check CRC
  - File system APIs
- Bagle tries to fool emulation with seldom APIs

Operating System Simulation

- Simulate each API directly
- Our generic unpacker support more than 100 APIs

Operating System Simulation

```
1000BF88: RegisterWindowMessage(ArmReReadMessage);
1000BF94: PostMessageA
1000F4F7: GetCurrentProcessId
1000F57F: CreateFileA("\\.\SICE", ..., 3, ...)
1000F593: GetLastError -> 2
1000F57F: CreateFileA("\\.\NTICE", ..., 3, ...)
1000F593: GetLastError -> 2
1000F57F: CreateFileA("\\.\SIWDEBUG", ..., 3, ...)
1000F593: GetLastError -> 2
1000F57F: CreateFileA("\\.\SIWVID", ..., 3, ...)
1000F593: GetLastError -> 2
```

Operating System Simulation

```
100082BC: RegOpenKeyEx(80000000, CLSID\{C9DC10FD-
D921-13D1B2E4-0060975B8649}, ..);
100078C8: GetSystemTime
10008230: GetTempPath (return c:\temp\)
10008262: CreateFileA("c:\temp\58AB070C.TMP",
..., 3, ...)
100078C8: GetSystemTime
10008113: RegOpenKeyEx(80000002, Software\The
Silicon Realms Toolworks\Armadillo, ..);
1000812C:
RegQueryValueEx(0,{65EED8A09843E1F6},..);
10008143: RegCloseKey
```

Operating System Simulation

```
479576: LoadLibraryA(advapi32.dll)
479646: GetProcAddress(77DA0000,
100194A8->RegCloseKey)
479646: GetProcAddress(77DA0000,
100194CA->RegOpenKeyExA)
479646: GetProcAddress(77DA0000,
100194DA->RegSetValueExA)
479646: GetProcAddress(77DA0000,
100194EC->RegCreateKeyExA)
479576: LoadLibraryA(shell32.dll)
479646: GetProcAddress(5,
1001950C->ShellExecuteA)
```

Typical Problems

- Error tracing
- When to stop?
- Speed

Error Tracing

- What do you do if your emulation engine does not emulate correctly?
- Millions of instructions
  - Error tracing very complex
- Solution: automatically debug during emulation
- Comparison reveals most errors

Error Tracing

```
Load_Real_Exe();
Start_Debugger_For_Real_Exe();

while (!different)
{
   Emulate_One_Instruction();
   Execute_One_Instruction_With_Debugger();
   Compare_Emulation_With_Reality();
}
```

When to stop?

- When can we stop the emulation?
  - Undecidable
- Fallback: maximum time-out
- Heuristic checks

When to stop?

Possible heuristic rules

- Entry point signatures from standard entry points
- Stop at some API functions
- Relationship between emulated opcodes/simulated API functions

Speed

- Speed is the most important problem
- CPU simulation needs to be as fast as possible
- Measure speed on AMD64 3000+
- Measure speed on Athlon XP 1200

Speed

Usual algorithm I

```
while (true)
{
   Decode_Instruction(r.eip);
   Lookup_Simulation_Function();
   Execute_This_Instruction();
}
```

Speed

Usual algorithm 2

```
void Xor_Simulation_Function()
{
   Decode_Mod_Rm(op_1, op_2);
   if (op_2 is memory)
      ReMap_Memory(op_2);
   op_1 = op_1 XOR op_2;
}
```

Speed

Usual algorithm - Speed (UPX)

CPU	Instructions	Time(ms)	Speed (MIPS)
AMD64	5.939.677	859	6,9
Athlon XP	5.939.677	1317	4,5

Speed

- Code usually does not change
  - Fast decoding cache
- Generate special code at runtime
  - Reduces conditional jumps
- Form blocks and link them together
  - Reduces unpredictable jumps

Speed

Improved emulation algorithm I

```
while (true)
{
   if (Is_Instruction_In_Cache(r.eip))
      Jump_To_Instruction_Code();
   else
      Translate_Instruction();
}
```

Speed

Improved emulation algorithm 2

```
void Translate_Instruction()
{
   Decode_Instruction(r.eip);
   Generate_Individual_Code();
   Insert_Code_Into_Cache();
   Chain_Code_Block();
}
```

Speed

Improved algorithm - Speed (UPX) comparison

CPU	Instructions	Time(ms)	Speed (MIPS)
AMD64	5.939.677	67 (859)	89 (6,9)
Athlon XP	5.939.677	104 (1317)	57 (4,5)

Results

- Generic unpacking not often used in anti-virus engines
- Showed that generic unpacking is possible in practise

Results

- Armadillo, ASPack, Exegriper, FSG, Lamecrypt, Mew, Morphine, Neolite, Netwalker, PCShrink, PECompact, PEPack, Petite, PEX, PKLite32, StonesPECrypter, UPack, UPX, Winkript, YodaCrypter
- Can be unpacked in 20 - 300ms (80-100MIPS)
- All variants and modified versions

Recent improvements

- Improved code quality
- Performed compiler like optimization
- Improved perfomance:
  - Lamecrypt **486** MIPS (34 MIPS)
  - UPX **553** MIPS (89 MIPS)

Questions/Comments?

You can also write an e-mail to

tobias.graf@ewido.net