Virus Obfuscation

Wei Wang

Code Obfuscation

- **obfuscation** is the deliberate act of creating obfuscated code, i.e. source or machine code that is difficult for humans to understand.
- Virus employs obfuscation to defeat anti-virus software and human analysis
- Virus encryption is one type of obfuscation.
- More complex obfuscations: oligomorphic, polymorphic and metamorphic viruses

Oligomorphic, Polymorphic and Metamorphic Viruses

- Viruses that can evolve by mutating as they replicate can be classified in three categories, based on the degree of variety they produce:
  1. Oligomorphic viruses can produce a few dozen decryptors; they select one at random when replicating
  2. Polymorphic viruses dynamically generate code rearrangements and randomly insert junk instructions to produce millions of variants
  3. Metamorphic viruses apply polymorphic techniques to the entire virus body rather than just to a decryptor, so that one generation differs greatly from the previous generation; no encryption is even necessary to be classified as metamorphic

Oligomorphic Viruses

- Detecting encrypted viruses that have distinctive decryptors was too easy (in the opinion of virus writers!)
- It carried several dozen decryptors in its body as data; when replicating, it selected one at random, encrypted the virus body with it, and deposited the body and the decryptor in the target file
- Whale was the first oligomorphic virus

Oligomorphic Viruses cont'd

- Carrying the decryptors as data is a burden to the virus, making it larger
- Memorial was a Windows 95 oligomorphic virus that generated 96 different decryptors, choosing one at replication time
  - Detecting 96 different patterns is an impractical solution for virus scanners that must deal with thousands of viruses; pattern database size explosion would result

Detecting Oligomorphic Viruses

- Limited numbers of decryptors can still be detected with pattern-matching
- Emulation, debugging, or proprietary dynamic analyses are needed to produce the decrypted virus for analysis

Polymorphic Virus

- Whereas an oligomorphic virus might create dozens of decryptor variants during replication, a polymorphic virus creates millions of decryptors
- Pattern-based detection of oligomorphic viruses is difficult, but feasible
- Polymorphic virus insert junk instructions into its decryptor, making pattern-based detection of polymorphic viruses infeasible
- Amazingly, the first polymorphic virus was created for DOS in 1990, and called V2PX or 1260 (because it was only 1260 bytes!)

Junk Instructions

- A junk instruction can be a no-op or do-nothing instruction, but it can also be an instruction that uses registers or memory locations that are unused in the decryptor
- Given the following decryptor loop for the Memorial oligomorphic virus:

```
Decrypt:
 xor %al, (%esi)      ; decrypt a byte with key in AL
 inc %esi            ; go to next byte
 inc %al             ; slide the key up
 dec %ecx            ; decrement the byte counter
 jnz Decrypt         ; loop back if more to decrypt
```

Junk Instructions cont'd

- Code patterns can be obfuscated with junk instructions:

```
Decrypt:
 add %ebx, %edx          ; junk
 xor %al, (%esi)          ; decrypt a byte with key in AL
 dec %edx                ; junk
 inc %esi                ; go to next byte
 mov (whocares), %edx    ; junk
 inc %al                 ; slide the key up
 dec %ecx                ; decrement the byte counter
 jnz Decrypt             ; loop back if more to decrypt
```

Junk Instructions cont'd

- A different variant puts different junk instructions at different offsets:

```
Decrypt:
 add  $4, %bh            ; junk
 xor  %edx, %edx         ; junk
 xor  %al, (%esi)        ; decrypt a byte with key in AL
 inc  %esi               ; go to next byte
 xchg %edx, %ebx         ; junk
 inc  %al                ; slide the key up
 cmp  %ecx, %edx         ; junk
 dec  %ecx               ; decrement the byte counter
 jnz  Decrypt            ; loop back if more to decrypt
```

Instruction Variation

- The index increment instructions are order-independent, creating more variants:

```
Decrypt:
 add  $4, %bh          ; junk
 xor  %edx, %edx       ; junk
 xor  %al, (%esi)      ; decrypt a byte with key in AL
 inc  %al              ; slide the key up
 xchg %edx, %ebx       ; junk
 inc  %esi             ; go to next byte
 cmp  %ecx, %edx       ; junk
 dec  %ecx             ; decrement the byte counter
 jnz  Decrypt          ; loop back if more to decrypt
```

Instruction Variation cont'd

- There is more than one way to increment or decrement counters:

```
Decrypt:
 add  $4, %bh          ; junk
 xor  %edx, %edx       ; junk
 xor  %al, (%esi)      ; decrypt a byte with key in AL
 add  $1, %al          ; slide the key up
 xchg %edx, %ebx       ; junk
 add  $1, %esi         ; go to next byte
 cmp  %ecx, %edx       ; junk
 sub  $1, %ecx         ; decrement the byte counter
 jnz  Decrypt          ; loop back if more to decrypt
```

Instruction Variation cont'd

- There is more than one way to increment or decrement counter and loop back:

```
Decrypt:
 add  $4, %bh            ; junk
 xor  %edx, %edx         ; junk
 xor  %al, (%esi)        ; decrypt a byte with key in AL
 add  $1, %al            ; slide the key up
 xchg %edx, %ebx         ; junk
 add  $1, %esi           ; go to next byte
 cmp  %ecx, %edx         ; junk
 sub  $1, %ecx           ; decrement the byte counter
 loop Decrypt            ; loop back if more to decrypt;ECX is
                         ; automatically decremented and
                         ; checked by loop instruction
```

Polymorphic Virus Example: The 1260 Virus

- A researcher, Mark Washburn, wanted to demonstrate to the anti-virus community that string-based scanners were not sufficient to identify viruses
- Washburn wanted to keep the virus compact, so he:
  - Modified the existing Vienna virus
  - Limited junk instructions to 39 bytes
  - Made the decryptor code easy to reorder

The 1260 Virus Decryptor

```
; Group 1: Prologue instructions
                    ; optional junk instruction slot 1
  mov 0x0e9b, %ax   ; set key 1
                    ; optional junk instruction slot 2
  mov 0x012a, %di   ; offset of virus Start
                    ; optional junk instruction slot 3
  mov 0x0571, %cx   ; byte count, used as key 2

; Group 2: Decryption instructions
Decrypt:
                    ; optional junk instruction slot 4
  xor %cx, (%di)    ; decrypt first 16-bit word with key 2
                    ; optional junk instruction slot 5
  xor %ax, (%di)    ; decrypt first 16-bit word with key 1
                    ; optional junk instruction slot 6

; Group 3: Decryption instructions
                    ; optional junk instruction slot 7
  inc %di           ; move on to next byte
                    ; optional junk instruction slot 8
  inc %ax           ; slide key 1
                    ; optional junk instruction slot 9

; loop instruction (not part of Group 3)
  loop Decrypt      ; slide key 2 (CX) and loop back if not zero

; Random padding up to 39 bytes
Virus:              ; encrypted virus body starts here
```

The 1260 Virus: Polymorphism

- Sources of decryptor diversity:
  - Reordering instructions within groups
  - Choosing junk instruction locations
  - Changing which junk instructions are used
- We will see that these variations are simple for the replication code to produce
- We see that it can really produce millions of variants in a short decryptor, just using these simple forms of diversity.

1260 Polymorphism: Reordering

- The 1260 decryptor has three instruction groups, with 3, 2, and 2 instructions, respectively
- The groups were defined to be the instruction sequences that could be permuted without changing the result of the decryption
  - i.e. there is no inter-instruction dependence among the instructions inside a group
- So, the reorderings within the groups produce 3! * 2! * 2! = 24 variants
- This gives a multiplicative factor of 24 to apply to all variants that can be produced using junk instructions

1260 Polymorphism: Junk Locations

- In a 2-instruction group, there are three locations for junk: before, after, and in between the two instructions
- However, there are far more possibilities than these three locations, as each location can hold many instructions
  - 39-byte junk instruction limit (imposed by virus designer)
  - Shortest x86 instructions take one byte; most take 2-3 bytes
- Conservatively, we could say that the replicator will choose about 15 junk instructions that will add up to 39 bytes
- 9 locations are possible throughout the decryptor
- This gives several thousands of possible ways of inserting junk instructions

1260 Polymorphism: Junk Instruction Selection

- How many instructions qualify as junk instruction candidates for this decryptor?
- The x86 has more than 100 instructions
- Each has dozens of variants based on operand choice, register renaming, etc.:
  - add %ax,%bx add %bx,%ax add %dx,%cx add %ah,%al etc...
  - Immediate operands produce a combinatorial explosion of possibilities
- Using only the registers that are unused by the decryptor will still produce hundreds of thousands of possibilities
- Combining all three polymorphisms: 24 * (several thousand) * (hundreds of thousands) of variants = ~1 billion variants

Simplified Polymorphism in 1260

- The 1260 virus made its replication code simpler by only allowing up to 5 junk instructions in any one location, and by generating only a few hundred of the possible x86 junk instructions
- That means it can produce a million or so variants rather than a billion
- A short (1260 byte) virus is still able to use polymorphism to achieve a million variants of the short decryptor code
- Pattern-based detection is hopeless

Polymorphism: Register Replacement

- The 1260 virus did not make use of another polymorphic technique: register replacement
- If the decryptor only uses three registers, the virus can choose different registers for different replications
- Another multiplicative factor of several dozen variants can be added by this technique
  - A decryptor of only 8 instructions can produce over 100 billion variants by the fairly simple application of four polymorphic techniques!

Polymorphic Mutation Engines

- Creating a polymorphic virus that makes no errors in replication and always produces functional offspring is difficult for the average virus writer
- Early in the history of virus polymorphism, a few virus writers started creating mutation engines, which can transform an encrypted virus into a polymorphic virus
- The Dark Avenger mutation engine, also called MtE, was the first such engine (DOS viruses, summer 1991, from Bulgaria)

Mutation Engine Example: MtE

- MtE was a modular design that accepted various size and target file location parameters, a virus, a decryptor, a pointer to the virus code to encrypt, a pointer to a buffer to write its output into, and a bit mask telling it what registers to avoid using
- The engine then generated the polymorphic wrapper code to surround the virus code and replicate it polymorphically
- MtE relied on generating variants of code obfuscation sequences in the decryptor, in additional to inserting junk instructions
  - There are many convoluted ways to compute any given number

MtE Example

- The following code is generated by MtE that sets BP to 0x0d2b

```
mov $0xA16C, %bp
mov $0x03,   %cl
ror %cl,     %bp
mov %bp,     %cx     ; Save 1st mystery value in cx
mov $0x856e, %bp
or  $0x740f, %bp
mov %bp,%si          ; Save 2nd mystery value in si
mov $0x3b92, %bp     ; Put 3rd value into bp
add %si,%bp          ; bp := bp+ 2nd mystery value
xor %cx,%bp          ; xor result with 1st mystery value
sub $0xb10c, %bp     ; BP now has the desired value
```

- Many different obfuscated sequences can compute the same value into BP

Detecting Polymorphic Viruses

- Anti-virus scanners in 1990-1991 were unable to cope, at first, with polymorphic viruses
- Soon, x86 virtual machines (emulators) were added to the scanners to emulate short stretches of code to determine if the result of the computations matched known decryptors
- This spurred the development of the anti-emulation techniques used in armored viruses

Detecting Polymorphic Viruses cont'd

- The key to detection is that the virus code must be decrypted to plain text at some point
- However, this implies that dynamic analysis must be used, rather than static analysis, and anti-emulation techniques might inhibit the most widely used dynamic analysis technique
  - Some polymorphic viruses combine EPO techniques with anti-emulation techniques
- An SDT might be executed up to the point of decryption; then the virus body can be examined in the SDT memory or dumped by the instrumentation

Metamorphic Viruses

- A metamorphic virus has been defined as a body-polymorphic virus; that is, polymorphic techniques are used to mutate the virus body, not just a decryptor
- Metamorphism makes the virus body a moving target for analysis as it propagates around the world
- The techniques used to transform virus bodies range from simple to complex

Metamorphism: Source Code

- Unix/Linux systems almost always have a C compiler installed and accessible to all users
- A source code metamorphic virus such as Apparition injects source code junk instructions into a C-language virus and invokes the C compiler
- By using junk variables at the source code level, the bugs that afflict many polymorphic and metamorphic viruses at the ASM level (e.g. accidentally using a register that is implicitly used by another instruction and was not really available for junk code) are avoided
- Because of differences in compiler versions, compiler libraries, etc., the resulting executable could vary across systems even if there were no source code metamorphism

.NET/MSIL Metamorphism

- Windows systems do not always have a C compiler available
- Windows systems with some release of Microsoft .NET installed will compile MSIL (Microsoft Intermediate Language) into the native code for that machine
- A source code metamorphic virus can operate on MSIL code and invoke the .NET Framework to compile it
- The MSIL/Gastropod virus is one example

Metamorphism: Register Replacement

- Regswap was a Windows 95 metamorphic virus released in December, 1998
- The metamorphism was restricted to register replacement, as in these two generations:

```
pop edx
mov edi,0004h
mov esi,ebp
mov eax,000Ch
add edx,0088h
mov ebx,[edx]
mov [esi+eax*4+1118],ebx
```

```
pop eax
mov ebx,0004h
mov edx,ebp
mov edi,000Ch
add eax,0088h
mov esi,[eax]
mov [edx+edi*4+1118],esi
```

Detecting Regswap

- Register replacement is not much of an obstacle to a hex-pattern scanner that allows the use of wild cards (don't-cares) in its patterns:
- The first two instructions of the previous example, in hex (machine code), are:

```
5A
BF04000000
```

```
5B
BB04000000
```

- Only the hex digits that encode registers differ
- If the scanner accepts wild cards, then both variants match 5?B?04000000

Metamorphism: Module Permutation

- Another metamorphism of the virus body is to reorder the modules
- Works best if code is written in many small modules
- First used in DOS viruses that did not even use encryption of the virus body, as a technique to defeat early scanners
- 8 modules produce 8! = 40,320 permutations; however, short search strings (within modules) can still work if wild cards are used to mask the particular addresses and offsets in the code

Metamorphism: Instruction Permutation

- The Zperm virus family used a method known from a DOS virus: reorder individual instructions and insert jumps to retain the code functionality
- Look at 3 generations of Zperm pseudocode:

```
jmp instr1
instr 4
instr 5
jmp END
instr 1
instr 2
jmp instr3
Instr 3
junk instr
jump instr4
END:
```

```
jmp instr1
instr 2
jmp instr3
instr 1
jmp instr2
instr 3
junk instr
Instr4
junk instr
instr5
END:
```

```
jmp instr1
instr 3
instr 4
jmp instr5
junk instr
instr 5
jmp END
instr1
instr2
jump instr3
END:
```

Instruction Permutation Detection

- Algorithmic Scanning
  - Virus-specific detection algorithms
- A Phoenix Analysis Tool, or an SDT, could make use of existing compiler transformations to simplify the jump chain into straight-line code
- If the virus used no other metamorphic technique besides permutation, it could then be recognized by patterns
  - However, Zperm and related viruses also use instruction replacement, junk instruction insertion, etc. to be truly metamorphic even after jump chains are straightened

Metamorphism: Build-and-Execute

- The Zmorph metamorphic virus appeared in early 2000 with a unique approach
- Many small virus code subroutines are added at the end of a PE file
  - They form a call chain among themselves
  - Each is body-polymorphic (metamorphic)
  - Each builds a little virus code on the stack
  - Execution is then transferred to the stack area when the building is complete
  - Payload is not visible inside the virus in normal patterns for a scanner

Metamorphic Engines

- A metamorphic engine is a code replicator that has evolutionary heuristics built in:
  - Change arithmetic and load-store instructions to equivalent instructions
  - Insert junk instructions
  - Reorder instructions
  - Change built-in constants to computed values
- Built-in constants are particularly important to pattern-based scanners, so a metamorphic engine that can mutate constants from one generation to the next makes pattern-based static analysis difficult or impossible

Metamorphic Engine Example

- The following three versions of code are generated by a metamorphic engine

```
; generation 1
mov  dword ptr [esi],55000000h
mov  dword ptr [esi+0004],5151EC8Bh
```

```
; generation 2
mov  edi,55000000h      ; 2nd gen., constant not changed yet
mov  dword ptr [esi],edi
pop  edi                ; junk
push edx                ; junk
mov  dh,40h             ; junk
mov  edx,5151EC8Bh      ; constant not changed yet
push ebx                ; junk
mov  ebx,edx
mov  dword ptr [esi+0004],ebx
```

Metamorphic Engine Example cont'd

```
; generation 3
mov  ebx,5500000Fh      ; 3rd gen., constant has not changed
mov  dword ptr [esi],ebx
pop  ebx                ; junk
push ecx                ; junk
mov  ecx,5FC0000CBh     ; constant has changed
add  ecx, F191EBC0h     ; ECX now has original constant value
mov  dword ptr [esi+0004],ecx
```

- As it replicates, the metamorphic engine makes just a few changes each generation, but the AV scanner code patterns change drastically
- Eventually, all constants will be mutated many times

The General Weakness of Metamorphic Viruses

- In order to mutate their code generation after generation, metamorphic viruses have to re-analyze the mutated code that it generates.
- Thus, metamorphic viruses need to use some coding conversions, or develop special algorithms that will help them to detect their own obfuscations.
  - This means that there is a pattern within the mutation.
- Once this pattern is discovered by anti-virus researchers, it can then be detected.
  - Algorithmic detection with a virus-specific algorithm to extract vital instructions from the mutated virus body