Encrypted Viruses

Wei Wang

Encrypted Viruses

- Virus encryption for the reasons of:
  - Anti-disassemble; analysis-resistant
  - Anti-detection; code-pattern detection resistant
- Encryption takes many forms
- The most advanced, difficult to defeat viruses use encryption techniques
- We should understand detecting, and disinfecting various encrypted viruses

Simple Encryption

- Earliest encrypted viruses use very simple encryption and decryption algorithms
  - e.g. XORing code with its own address
- Goal:
  - NOT to use advanced algorithms that were hard to analyze
  - just to slow down analysis and defeat pattern-based virus detection
- Because the decryption code is always present in unencrypted form, there is not much point in choosing complex encryption/decryption methods
- The DOS virus Cascade was the first encrypted virus

Simple Encryption Example

- Based on a real virus: Cascade
- XOR is used to encrypt and decrypt
- XOR is reversible:
  - 0xf247 xor 0x0682 = 0x0f4c5
  - 0xf4c5 xor 0x0682 = 0x0f247
- Due to the speed and reversibility, XOR is a very good encryptor and decryptor

Simple Encryption Example cont'd

- The goal is to encrypt the following code (4bytes):

```
5e      pop %esi
5f      pop %edi
C9      leave
C3      ret
```

- Assume the code is injected to address @ 0x08084044
- Encryption algorithm
  - Code xor Address xor length_of_code = encrypted_code
  - 0xc3c95f5e xor 0x08084044 xor 0x00000004 = 0xcbc11f1e

Simple Encryption Example cont'd

- Decrypting algorithm:
  - encrypted_code xor length_of_code xor Address = code
  - 0xcbc11f1e xor 0x00000004 xor 0x08084044 = 0xc3c95f5e

Simple Encryption Example cont'd

- The simple decryptor:

```
 push %eax           ; save current EAX
 mov  %esp, %eax     ; save ESP into EAX
 lea  Virus,%esi     ; start of encrypted code (computed by virus)
 mov  $0x4, %esp     ; length of encrypted code (4 bytes)

Decrypt:
 xor  %esp,(%esi)    ; xor code with its address
 xor  %esi,(%esi)    ; xor code with its inverse index

 mov  %eax, %esp     ; restore ESP
 pop  %eax           ; restore EAX

Virus:               ; encrypted virus code body @ 0x08084044
 1e 1f c1 cb
```

Simple Encryption Example cont'd: More Virus Code

- The simple decryptor:

```
 push %eax          ; save current EAX
 mov  %esp, %eax    ; save ESP into EAX
 lea  Virus,%esi    ; start of encrypted code (computed by virus)
 mov  $0x684, %esp  ; length of encrypted code (1668 bytes)

Decrypt:
 xor  %esp,(%esi)   ; xor code with its address
 xor  %esi,(%esi)   ; xor code with its inverse index
 add  $4, %esi      ; increase esi to read next 4-byte code
 sub  $4, %esp      ; decrease ESP/code length by 4 bytes
 jnz  Decrypt       ; jump back to Decrypt if ESP is not 0

 mov  %eax, %esp    ; restore ESP
 pop  %eax          ; restore EAX

Virus:              ; encrypted virus code body
 … … … … … … … … …
 … … … … … … … … …
 … … … … … … … … …
```

Simple Encryption Example cont'd

- Summary of encrypt steps:
  - Find a cavity (the address of a cavity) to insert virus code
  - Encrypt the virus code based on the cavity address and virus code length
  - Inject encrypted virus code into the selected cavity
- Summary of decrypt steps:
  - Load the address of the virus code into ESI
  - Load the length of the virus code into ESP
  - Decrypted based on ESI and ESP values, and write the virus back to the cavity

Simple Encryption Example cont'd

- Very fast to encrypt and decrypt, yet sufficient to prevent detection by patterns
- IMPORTANT: Even the hex patterns are file-dependent, because they depend on addresses
- Why use ESP to save virus length?
  - With pattern-based detection impeded by encryption, an anti-virus researcher would like to step through the decryptor in a debugger and see the decrypted code
  - However, use of stack pointer inhibits most debugger use
  - Unfortunately, the code “mov virus_length, $esp” itself is a distinctive pattern (signature) for the Cascade virus

Analyzing Simple Encryption Virus

- Prevention in the OS: don’t allow writing to the executable code segment
  - work around 1: decrypting into a buffer on stack or heap, rather than decrypting code in its place
  - Work around 2: change the flag of .text section to be writable
- The best attack upon a simple encrypted virus is to detect the code patterns of the decryptor, e.g.
  - E,g., mov 0x0684, %esp ; length of encrypted code (1666 bytes)

Difficult Encryption

- Example 1: Two encryptors and two decryptors
  - Encrypt
    - One encryptor encrypts the virus body with one encryption algorithm
    - A second encryptor encrypts the virus body with another encryption algorithm in reverse order
  - Decrypt:
    - Requires to decryptors and two rounds of decryptions
  - Decryptors can still be detected

Difficult Encryption cont'd

- Example 2: Two encryptors and two decryptors
  - Encrypt:
    - The first encryptor encrypts the second decryptor code
    - The second encryptor encrypts the virus code
  - Decrypt:
    - The first decryptor decrypts the second decryptor
    - The second decryptor decrypts the virus code
  - Static analysis of the patterns of the first decryptor would be irrelevant; that decryptor could be common to many viruses and also to commercial software

Detecting Encrypted Viruses

- Because virus code is encrypted, the best way to detect these viruses is to detect the decryptor
- Indicator of a decryptor: tight loops with xors
  - But, many different viruses can use the same decryptor algorithm and have totally different payloads and behaviors
- Indicator of a decryptor: unique virus length
  - A virus could pad itself out so that it has the same length as other, unrelated viruses
- Even worse is the fact that some commercial software is obfuscated by an anti-debug wrapper, which looks just like the decryptor code for Cascade, in order to prevent reverse engineering of their product
  - Can produce false positives

Detecting Encrypted Viruses cont'd

- Because OS may override section flags and disallow direct write to text section, encrypted viruses have to allocate memory on stack or heap for its code
  - Can DEP prevent these viruses from execution? (answer in the last slide)
  - Allocating space on heap requires unencrypted allocation code, which is easier to detect – memory allocation along with some typical decrypting code can produce a good code pattern to match
  - Allocating space on stack is the stealthiest – a “sub $length %esp” is enough

Detecting Encrypted Viruses cont'd

- How can an encrypted virus be detected if it uses stack allocation, makes itself look like a commercial anti-debug wrapper, makes itself the same length as unrelated viruses, etc.?
- Emulation and dynamic analysis are common approaches
  - Expensive
  - Proprietary

Alternatives: Static Analysis and SDT

- An instrumentation tool could be generated to dump the code after the decryption has occurred
  - Still need to black-box or sandbox the application to prevent damage
- SDT (software dynamic translation, or run-time compilation) decodes a program into a buffer as it runs
  - Can examine decrypted code in the translation cache/buffer

Virus Code Evolution

- Simile is one example of a virus that evolves in order to frustrate pattern-based detection
- Each time it replicates, it generates a different memory allocation code sequence in the decryptor
  - Can be done with simple obfuscations, code re-orderings, etc.
  - No single pattern matches the allocator – avoid detection by memory allocator code
- More common is mutating the decryptor code itself and using stack allocation

Decryptor Mutation

- 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

Can DEP Prevent Execute Virus Code in Stack/Heap

- No. The OS allows user code to change the executable flag of a memory page. Virus decryptor can include a system call to change enable the executable flag of the memory pages for stack/heap
- The key difference between virus and buffer overflow code injection is: part or all of virus code (e.g., decryptor) is injected into the text section of a file, so this part of code will always be executed; buffer overflow code injection injects all of the code to stack/heap, so DEP can prevent their execution.