Encrypted and Oligomorphic Viruses
Encrypted Viruses

- Virus encryption is both
  - an anti-disassembly technique and
  - an obstacle to virus detection using code patterns
- Encryption takes many forms
- The most advanced, difficult-to-defeat viruses use encryption techniques
- We will devote several lectures to understanding, detecting, and disinfecting various encrypted viruses
- This is the first part of Chapter 7 of Szor.
Simple Encryption

- The earliest viruses to use encryption used a very simple decryption algorithm, such as XORing code with its own address.

- The point was **not** to use advanced algorithms that were hard to analyze;
  - just to slow down analysis and
  - defeat pattern-based virus detection.

- Decrypter code always present in unencrypted form,
  - not much point in choosing complex encryption/decryption methods.

- The DOS virus Cascade was the first encrypted virus.
Example: Cascade Virus

The simple decryptor of Cascade, circa 1990:

```assembly
lea   si,Start   ; start of encrypted code
        ; (computed by virus)
mov   sp,0682h   ; length of encrypted code (1666 bytes)
Decrypt:
    xor  [si],si   ; xor code with its address
    xor  [si],sp   ; xor code with its inverse index
    inc  si       ; increment address pointer
    dec  sp       ; decrement byte counter
    jnz  Decrypt  ; loop if more bytes to decrypt
Start:   ; virus code body
```
Cascade Virus Walkthrough

- Setting up the indices:
  
  ```
  lea si, Start
  ; start of encrypted code (computed by virus)
  ```

- The virus does not have a “Start” label whose address is determined by a compiler
  - Instead, it computes the address at infection time, depending on the location in the file being infected

- Virus uses hex offsets; we show “Start” to make it more readable
Cascade Virus Walkthrough

- Stack pointer used as counter
  
  ```
  mov  sp,0682h ; length of encrypted code (1666 bytes)
  ```

- Virus knows its own length before it infects a new file

- Using the stack pointer is an anti-debugger technique
  - Cascade is therefore an armored virus

- However, this line of code is a distinctive pattern (signature) for this virus
Cascade Virus Walkthrough

- The XOR encryption lines:
  xor [si],si ; xor code with its address
  xor [si],sp ; xor code with its inverse index

- The XOR operation is reversible:
  0f237h XOR 0682h = 0f4b5h
  0f4b5h XOR 0682h = 0f237h

- 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
Cascade Virus Walkthrough

- **Increment counters/indices and loop:**
  
  ```
  inc si ; increment address pointer
  dec sp ; decrement byte counter
  jnz Decrypt ; loop if more bytes to decrypt
  ```

- 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.
Analyzing Cascade

- Prevention in the OS: don’t allow writing to the executable code segment
  - Virus writer can work around this by decrypting into a buffer, rather than decrypting code in its place
- The best attack upon a simple encrypted virus is to detect the code patterns of the decryptor, e.g.
  \[
  \text{mov} \ sp, 0682h \; ; \text{length of encrypted code (1666 bytes)}
  \]
Difficult Decryptors

- One decryptor loop might traverse the virus body, applying a decryptor function (e.g. XOR or something more complex),
  - then another decryptor loop can traverse the virus code in reverse order applying a different decryption function, etc.

- Unencrypted decryptor code could:
  - decrypt a piece of code that is a more complex decryptor,
  - …which then decrypts another decryptor,
  - …which decrypts the virus

- 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
  - i.e., first decryptor is legitimate, commonly used decryptor
Decryptor loop examples

- Decryptor
- Decryptor
- Decryptor

Decryptor 1
- Decryptor 2
- Decryptor 3
Decryptor strategies (cont’d)

- Change decryption direction
- Multiple layers of encryption
- Mixed directions
- One decryptor/Multiple keys
- Obfuscate Decryptor start (EPO) with padding, etc.
- Non-linear decryption
Detecting Decryptors

- The main loop of the decryptor (a tight loop with XORs) looks like it would be a good subject for pattern-based detection
  - But, many different viruses can use the same decryptor algorithm and have totally different payloads and behaviors
- A virus could pad itself out so that it has the same length as other, unrelated viruses – “mimicry”
- **Doh!** 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 Decryptors cont.

- Memory allocation within the decryptor can produce a good code pattern to match
- Decryptor has three locations in which it can decrypt the virus code:
  1. **In place;** OS can disallow this
  2. **In heap;** allocation code is unencrypted and makes pattern-based detection easier
  3. **On the stack;** stealthiest choice --- why?
Detecting Decryptors cont.

- 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
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
- More common is mutating the decryptor code itself and using stack allocation
- We’ll have more to say about Simile when we discuss Metamorphism.
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
   1. polymorphic techniques to the entire virus body rather than just to a decryptor, so that
   2. one generation differs greatly from the previous generation;
   3. 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!)
- Whale was the first oligomorphic virus
- 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
Oligomorphic Viruses cont.

- 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
- Memorial inserted junk instructions at various points in the decryptor code
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:

```asm
Decrypt:
    xor [esi], al ; 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
```
Code patterns can be obfuscated with junk instructions:

Decrypt:

```
add ebx, edx ; junk
xor [esi], al ; 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
```
A different variant puts different junk instructions at different offsets:

Decrypt:
```
add bh, 4           ; junk
xor edx, edx        ; junk
xor [esi], al       ; decrypt a byte with key in AL
inc esi            ; go to next byte
xchg ebx, edx       ; junk
inc al             ; slide the key up
cmp ebx, edx        ; junk
dec ecx            ; decrement the byte counter
jnz Decrypt        ; loop back if more to decrypt
```
The index increment instructions are order-independent, creating more variants:

Decrypt:

```
add bh,4             ; junk
xor edx,edx          ; junk
xor [esi],al         ; decrypt a byte with key in AL
inc al               ; slide the key up
inc esi              ; junk
inc esi              ; go to next byte
cmp ebx,edx          ; junk
dec ecx              ; decrement the byte counter
jnz Decrypt          ; loop back if more to decrypt
```
There is more than one way to increment or decrement counters:

Decrypt:

```assembly
add bh, 4           ; junk
xor edx, edx        ; junk
xor [esi], al       ; decrypt a byte with key in AL
add al, 1           ; slide the key up
xchg ebx, edx       ; junk
add esi, 1          ; go to next byte
cmp ebx, edx        ; junk
sub ecx, 1          ; decrement the byte counter
jnz Decrypt         ; loop back if more to decrypt
```
Junk Instructions cont.

- There is more than one way to decrement a counter and loop back if it is not zero:

Decrypt:

```assembly
add bh,4           ; junk
xor edx,edx        ; junk
xor [esi],al       ; decrypt a byte with key in AL
add al,1           ; slide the key up
xchg ebx,edx       ; junk
add esi,1          ; go to next byte
cmp ebx,edx        ; junk
loop Decrypt       ; decrement the byte counter and
                    ; loop back if more to decrypt
```


Detecting Oligomorphic Viruses

- Clearly, it is easy to produce numerous variants of a decryptor.
- Filtering out no-ops and do-nothings does not remove the obfuscation.
- Emulation, debugging, or proprietary dynamic analyses are needed to produce the decrypted virus for analysis.