Anti-Anti-Virus Techniques

CS4001/7001 Malware Analysis & Defense

Bill Harrison

Defeating Virus Detection

- Although disassembly and human inspection of code will always have a place, virus writers can make such inspection very difficult:
  - Creating viruses in obfuscated assembly code
  - Encrypting virus code
- Encryption issues will be examined in detail over the next several weeks
- **Code obfuscation techniques** are available in assembly language programs that make code tedious to read and understand

Obfuscated Code Example

- Can you understand what this 16-bit DOS code is doing?

```
mov ax,0fe05h
jmp $-2
sub ax,9e05h
push ax
sub ah,43h
```

1. The jump goes back 2 bytes relative to its own position. This puts it in the middle of the previous instruction!

Obfuscated Code Example cont'd.

```
mov ax,0fe05h    →    0fe05h jmp (mov ah, …)
jmp $-2          →    $-2
sub ax,9e05h     →    sub ax,9305h
push ax          →    push ax
sub ah,43h       →    sub ah,43h
```

2. 0fe05h is now interpreted as an opcode, which is of the form **`mov ah,operand2`**, where **`operand2`** is taken from the **`jmp`** opcode.
3. The `$-2` operand is now interpreted as a **`cld`** instruction (clear direction flag), basically a no-op

Obfuscated Code Example cont'd.

```
mov ax,0fe05h        →    0fe05h jmp (mov ah, …)
jmp $-2              →    $-2
sub ax,9e05h         →    sub ax,9305h
push ax              →    push ax
sub ah,43h           →    sub ah,43h
```

4. That leaves the instruction pointer perfectly positioned on the `sub ax,9e05h` instruction.
   - The `ax` register no longer has `0fe05h` in it; because of the `mov ah,operand2` instruction, the value `0ea05h` is in register `ax`
5. Subtraction: `0ea05h` minus `9e05h` gives `4c00h`

Obfuscated Code Example cont'd.

```
mov ax,0fe05h          →    0fe05h jmp (mov ah, …)
jmp $-2                →    $-2
sub ax,9e05h           →    sub ax,9305h
push ax                →    push ax
sub ah,43h             →    sub ah,43h
```

6. The value `4c00h` is pushed on the stack.
   - This happens to be the address of the function DOS uses to terminate program execution; the virus will use it later
7. In register `ah, 4ch` minus `43h` gives `09h` , which happens to be the DOS Print String function,
   - to be used soon to overwrite the stack, etc., etc. (details omitted for legal reasons!)

Detecting Obfuscated Viruses

- Some obfuscated assembly code is even designed to trash the program stack ONLY when the code is examined line by line in a debugger; this is an attack upon anti-virus researchers doing manual code inspections
- Clearly, once certain code themes have been identified by difficult inspection techniques,
  - anti-virus software needs a way to detect obfuscated versions, minor variants, etc., without manual disassembly and inspection of each executable file
- Automatic pattern matching tools provide a partial solution

Detecting Viruses with Patterns

- Early anti-virus software relied upon databases of file sizes
  - Easily defeated by stealth viruses that carefully leave the file size unchanged, e.g. compressors, cavity viruses, etc.
- DBs can be hacked, too.
  - Checksum databases can be attacked or removed by some viruses, as can file size databases
- Scanning executable files to detect virus code became necessary early in the ongoing battle between virus and anti-virus software designers

Detecting Viruses with Patterns

- Some Patterns are Suspicious.
  - As anti-virus software engineers tediously disassembled and studied viruses by hand, they noticed certain code sequences that legitimate application code would be unlikely to contain
    - Writing to the IVT (interrupt vector table)
    - Calling the system function to reduce available memory and then writing to the dead spot created
    - Writing to the PT (partition table) entries in the MBR
    - Executing a tricky jump
- Hexadecimal opcode sequences were extracted from viruses and placed in a pattern file
- Simple hexadecimal scanners examined binaries to find these opcode sequences

Defeating Simple Scanners

- Virus writers became expert at varying their code patterns
  - Some opcode sequences could be **reordered** slightly without affecting functionality
  - **Random No-ops** could be inserted in the virus
  - A constant could be placed in the code, never to be used, and altered each time the virus replicated
- Virus generator kits could be designed to inject different types of obfuscation into different virus variants

Evolving Scanners: enriching the pattern language

- Binary scanners were enhanced to allow wildcard specifiers
  - E.g. Instead of looking for 0400 55F4 BB03, it might be better to look for 0400 55?? BB03, where ?? specifies a don't-care byte, because the F4 byte might be subject to variation
  - A better pattern could be 0400 55?? %3 BB03, which specifies that BB can be found anywhere in the next 3 bytes; if no-ops, or instructions that are effectively no-ops, are inserted, they do not prevent detection
  - But what if the intervening bytes are NOT effectively no-ops? These early scanners produced many false positives, which are costly to customers who unnecessarily restored files and lost recent work

Second Generation Scanners

- **Smart scanning** is a technique based on the observation that certain sections of the virus body made no references to data constants that might change, had no jump or call offsets that would change if no-ops were inserted, etc.
- The smart scanner looks only in such areas of the virus body, and skips over no-ops and do-nothings (e.g. `add eax,0`)

Regular Expression Matching

- **Regular expressions** can be built up from components that match the sections of a virus least able to be obfuscated, as with smart scanners
- Patterns for no-ops and do-nothings can be specified and used to prevent simple obfuscations
- Don't-care bytes are easy to specify
- A "regex" eases working with disassembled code (easier to read than hexadecimal)

Regex Matching cont'd

- Example:

```
^[ \t]*push offset.*\n[ \t]*pop eax\n[ \t]*jmp \[eax\]
```

- Reading from left to right: "From beginning of a line, match white space followed by push offset … end-of-line, then white space followed by pop eax end-of-line, then white space followed by jmp [eax]"
- Detects an obfuscated tricky jump:

```
push offset virus_func
pop eax
jmp [eax]
```

- ^ anchors pattern to the beginning of a line

Regex Matching cont'd

- We could complicate the regex a little to allow any number of no-ops in between the instructions we are matching:

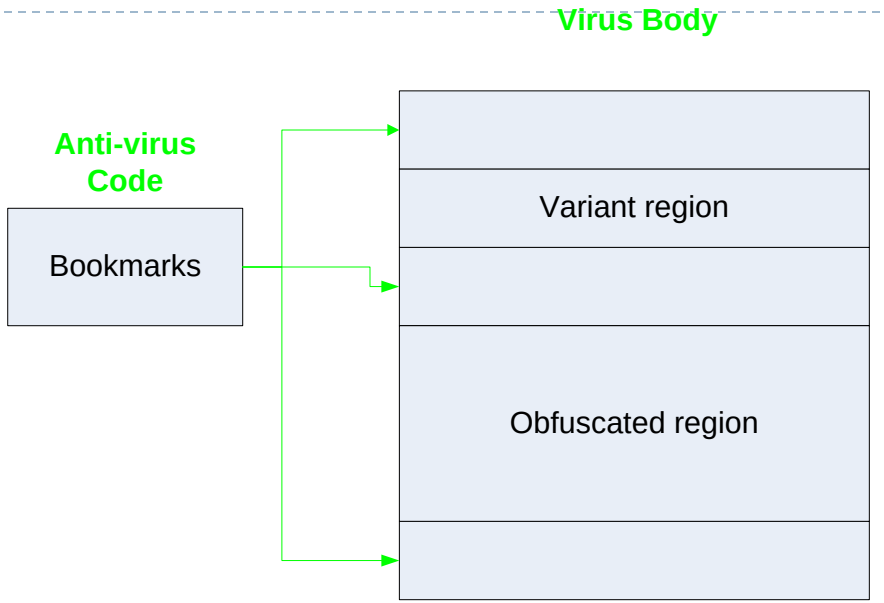
```
 ^[ \t]*push offset.*\n([ \t]*nop\n)*[ \t]*pop eax\n([
\t]*nop\n)* [ \t]*jmp \[eax\]
```

- A regular-expression package or scripting language will permit us to define named patterns so we don't keep typing the no-op pattern over and over
- We can then extend the no-op pattern to include a variety of do-nothing instructions

Nearly Exact Identification

- Instead of trying to find a single sequence of bytes unique to a virus, a scanner can have a collection of ***bookmarks* specifying patterns at relative distances** in the virus body
- These distances and patterns are determined by examining all the variants within a known family of viruses and noticing the code sequences they have in common
  - Can skip over the regions that are being obfuscated in the different variants, defeating efforts to make variants look different from the original

Nearly Exact Identification



- Instead of hex code patterns, some of the bookmarks could specify checksums or hash functions over certain regions of the virus body
- Very few false positives or false negatives; hence, *nearly exact identification*

Exact Identification

- As with nearly exact identification, patterns, checksums, and hash functions are computed over more than one region of code
- The improvement here is that every code region that is constant across the whole family of viruses is part of the examination (more time consuming, but more exact)
- Total code size is also a bookmark used (along with all the patterns and checksums) to identify the family and its variants
- Particular variants can be identified recursively after the virus family has been identified
- Tedious to build the patterns, slow to scan a system