Polymorphic & Metamorphic Viruses

CS4440/7440 Spring 2015

Evolution of Polymorphic Viruses (1)

- Why polymorphism?
  - Anti-virus scanners detect viruses by looking for signatures (snippets of known virus code)
    - Virus writers constantly try to foil scanners
- Encrypted viruses: virus consists of a constant decryptor, followed by the encrypted virus body
  - Cascade (DOS), Mad (Win95), Zombie (Win95)
  - Relatively easy to detect because decryptor is constant
- Oligomorphic viruses: different versions of virus have different encryptions of the same body
  - Small number of decryptors (96 for Memorial viruses); to detect, must understand how they are generated

Evolution of Polymorphic Viruses (2)

- Polymorphic viruses: constantly create new random encryptions of the same virus body
  - Marburg (Win95), HPS (Win95), Coke (Win32)
  - Virus must contain a polymorphic engine for creating new keys and new encryptions of its body
    - Rather than use an explicit decryptor in each mutation, Crypto virus (Win32) decrypts its body by brute-force key search
- Polymorphic viruses can be detected by emulation
  - When analyzing an executable, scanner emulates CPU for a time.
    - Virus will eventually decrypt and try to execute its body, which will be recognized by scanner.
  - This only works because virus body is constant!

Anti-antivirus techniques

MOV A,R1	MOV A,R1	MOV A,R1	MOV A,R1	MOV A,R1
ADD B,R1	NOP	ADD #0,R1	OR R1,R1	TST R1
ADD C,R1	ADD B,R1	ADD B,R1	ADD B,R1	ADD C,R1
SUB #4,R1	NOP	OR R1,R1	MOV R1,R5	MOV R1,R5
MOV R1,X	ADD C,R1	ADD C,R1	ADD C,R1	ADD B,R1
	NOP	SHL #0,R1	SHL R1,0	CMP R2,R5
	SUB #4,R1	SUB #4,R1	SUB #4,R1	SUB #4,R1
	NOP	JMP .+1	ADD R5,R5	JMP .+1
	MOV R1,X	MOV R1,X	MOV R1,X	MOV R1,X
			MOV R5,Y	MOV R5,Y
(a)	(b)	(c)	(d)	(e)

- Examples of a polymorphic virus
  - Do all of these examples do the same thing?

Polymorphic Viruses

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

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
    - What's a junk instruction?
  - Made the decryptor code easy to reorder

The 1260 Virus Decryptor (single instance)

```
; Group 1: Prologue instructions
mov ax,0E9Bh        ; set key 1
mov di,012Ah        ; offset of virus Start
mov cx,0571h        ; byte count, used as key 2
; Group 2: Decryption instructions
Decrypt:
xor [di],cx         ; decrypt first 16-bit word with key 2
xor [di],ax         ; decrypt first 16-bit word with key 1
; Group 3: Decryption instructions
inc di              ; move on to next byte
inc ax              ; slide key 1
; loop instruction (not part of Group 3)
loop Decrypt        ; slide key 2 and loop back if not zero
; Random padding up to 39 bytes
Start:              ; encrypted virus body starts here
```

The 1260 Virus: Polymorphism

- Sources of decryptor diversity:
  1. Reordering instructions within groups
  2. Choosing junk instruction locations
  3. Changing which junk instructions are used
- These variations are simple for the replication code to produce
- Can we really produce millions of variants in a short decryptor, just using these simple forms of diversity?

Polymorphism: Reordering in 1260

- The 1260 decryptor has three instruction groups,
  - Each with 3, 2, and 2 instructions, respectively
  - Groups are instruction sequences that, when permuted, do not change decryption result
    - i.e. there is no inter-instruction dependence among the instructions inside a group
- 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

Polymorphism: Junk Locations in 1260

- In 2-instruction group, three locations for junk: before, after, and in between the two instructions
- Far more possibilities than these three locations,
  - each location can hold from zero to 39 instructions
    - 39-byte junk instruction limit
      - imposed by virus designer
    - Shortest x86 instructions take one byte; most take 2-3 bytes
    - Conservatively, assume replicator will choose about 15 junk instructions that will add up to 39 bytes
    - 11 locations are possible throughout the decryptor

Junk Locations in 1260 (cont' d)

- The choosing of 11 numbers from 0-15, that add up to exactly 15, can be done in how many ways?
  - `1+10+(10+C(10,2))+(10+P(10,2)+C(10,3)) +(10+P(10,2)+C(10,2)+10+C(9,2)+C(10,4))+......`

    `= 1+10+55+220+401+......`

    = *approx 3K ways*
- Multiplicative factor of several thousand to apply to all variants that can be produced using junk instruction selection and decryptor instruction reordering
  - So far, 24 * (several thousand) variants

Recall $C(n,k) = \frac{n!}{k!(n-k)!}$ $P(n,k) = \frac{n!}{(n-k)!}$.

Polymorphism: Junk Instruction Selection

- How many instructions qualify as junk instruction candidates for this decryptor?
- The x86 has more than 100 instruction varieties
- Each has dozens of variants based on operand choice, register renaming, etc.:
  - `add ax,bx` `add bx,ax` `add dx,cx` `add ah,al`
  - `add si,1` `add di,7` `etc.`
  - `Immediate operands produce a combinatorial explosion of possibilities`
- Using only registers unused by decryptor still produces hundreds of thousands of possibilities
  - 24 * (several thousand) * (hundreds of thousands) of variants = ~1 billion variants

Polymorphism in V2PX/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

Bottom Line: Pattern-based detection is hopeless

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!

Mutation Engines

- Creating a polymorphic virus is difficult
  - Must makes no errors in replication
  - Always produces functional offspring is
    - Beyond 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)

MtE Mutation Engine

- MtE was a modular design that accepted
  - various size and target file location parameters,
  - a virus body,
  - 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
- MtE 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, rather than inserting junk instructions
  - E.g., there are many ways to compute any given number

MtE Decryptor Obfuscation/Hiding the key

- Can you follow the computation of a value into register BP below?

```
mov bp,A16Ch
mov cl,03h
ror bp,cl
mov cx,bp             ; Save 1st mystery value in cx
mov bp,856Eh
or bp,740Fh
mov si,bp             ; Save 2nd mystery value in si
mov bp,3B92h          ; Put 3rd value into bp
add bp,si             ; bp := bp+ 2nd mystery value
xor bp,cx             ; xor result with 1st mystery value
sub bp,B10Ch          ; BP now has the desired value
```

- Many sequences compute the same value in BP

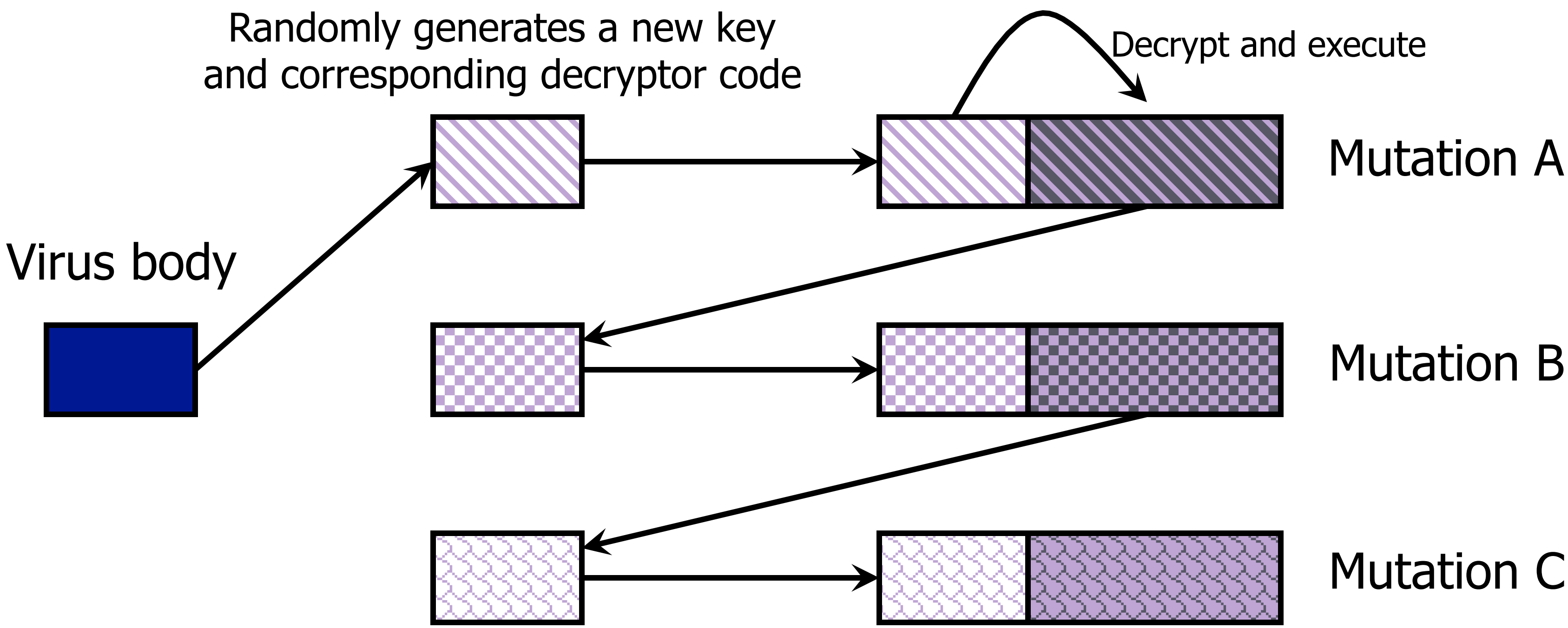
Detecting Polymorphic Viruses

- 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 symbolically evaluate 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

- 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
- Anti-emulation techniques might inhibit the most widely used dynamic analysis technique
  - E.g., Some polymorphic viruses combine EPO techniques with anti-emulation techniques
  - E.g., Use multiple encryption passes to obfuscate the virus body

Virus Detection by Code Emulation



To detect an unknown mutation of a known virus, emulate CPU execution of until the current sequence of instruction opcodes matches the known sequence for virus body

Today, next week, and the week after that.

- Reading assignment: "Hunting for Metamorphic" by Szor and Ferrie.
  - This is required reading.
- Wednesday the 8th: Jon Rolf of NSA will visit our class.
  - Jon is a 1988 graduate of MU ECE and he'll talk about his career with the agency
- Monday the 13th. Midterm.
  - Covers everything since the last quiz
  - Especially Chapter 7 and "Hunting for Metamorphic"
- Wednesday the 15th and Friday the 17th: Lecture cancelled
  - I'll be travelling
  - I will make an assignment in lieu of lecture.

Metamorphic Viruses

- Obvious next step: mutate the virus body, too!
- Virus can carry its source code (which deliberately contains some useless junk) and recompile itself
  - Apparition virus (Win32)
  - Virus first looks for an installed compiler
    - Unix machines have C compilers installed by default
  - Virus changes junk in its source and recompiles itself
    - New binary mutation looks completely different!
- Many macro and script viruses evolve and mutate their code
  - Macros/scripts are usually interpreted, not compiled

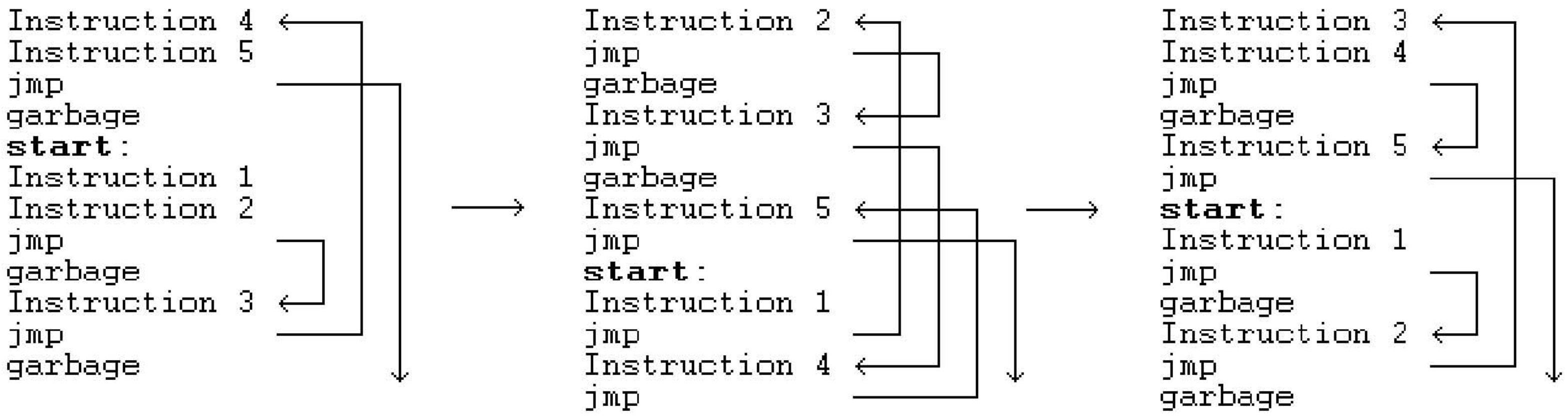
Metamorphic Mutation Techniques

- Same code, different register names
  - Regswap (Win32)
- Same code, different subroutine order
  - BadBoy (DOS), Ghost (Win32)
  - If n subroutines, then n! possible mutations
- Decrypt virus body instruction by instruction, push instructions on stack, insert and remove jumps, rebuild body on stack
  - Zmorph (Win95)
  - Can be detected by emulation because the rebuilt body has a constant instruction sequence

Real Permutating Engine (RPME)

- Introduced in Zperm virus (Win95) in 2000
- Available to all virus writers, employs entire bag of metamorphic and anti-emulation techniques
  - Instructions are reordered, branch conditions reversed
  - Jumps and NOPs inserted in random places
  - Garbage opcodes inserted in unreachable code areas
  - Instruction sequences replaced with other instructions that have the same effect, but different opcodes
    - Mutate SUB EAX, EAX into XOR EAX, EAX or
      PUSH EBP; MOV EBP, ESP into PUSH EBP; PUSH ESP; POP EBP
- Bottom Line: There is no constant, recognizable virus body!

Example of Zperm Mutation



- From Szor and Ferrie, "Hunting for Metamorphic"

Defeating Anti-Virus Emulators

- Recall: to detect polymorphic viruses, emulators execute suspect code for a little bit and look for opcode sequences of known virus bodies
- Some viruses use random code block insertion engines to defeat emulation
  - E.g., Routine inserts a code block containing millions of NOPs at the entry point prior to the main virus body
  - Emulator executes code for a while, does not see virus body and decides the code is benign… when main virus body is finally executed, virus propagates
  - Bistro (Win95) used this in combination with RPME

Putting It All Together: Zmist

- Zmist was designed in 2001 by Russian virus writer Z0mbie of "Total Zombification" fame
- New technique: code integration
  - Virus merges itself into the instruction flow of its host
  - "Islands" of code are integrated into random locations in the host program and linked by jumps
  - When/if virus code is run, it infects every available portable executable
    - Randomly inserted virus entry point may not be reached in a particular execution

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

Source Code Metamorphism

- 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
- Amateur virus writers often created buggy viruses when they attempted to use polymorphism.
  - Source code metamorphism is easier to do correctly.

.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
  - Probably a fertile field for viruses in the near future
- The MSIL/Gastropod virus is one example

Early Metamorphic Viruses

- Very few on DOS, but the first was a DOS virus called ACG (Amazing Code Generator)
- The code generator generated a new version of the virus body each time it replicated (thus it was metamorphic)
- Although most metamorphic viruses use encryption, ACG did not
  - Being "body-polymorphic" is sufficient to avoid pattern-based detection
- ACG was not too damaging, because DOS was already a dying operating system when it was released in 1997
- This is a key difference between polymorphic and metamorphic viruses: the former all mutate the decryptor, the latter might not even have a decryptor

Early Metamorphics: Regswap

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

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

Detecting Regswap

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

```
5A          58
BF04000000  BB04000000
```

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

Module Permutation

- Another metamorphosis 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

Metamorphic 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
- Emulators are used to detect Zmorph, as well as many other metamorphic viruses

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 Evol virus of July, 2000
- Compare a code snippet from two generations, after several generations of evolution:

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

Evol Example cont.

- A later generation shows the constant mutation starting:

```
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

Metamorphic 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 three generations of Zperm pseudocode:

```
jmp Start
Instr4
Instr5
jmp End
junk
Start:
Instr1
Instr2
jmp Instr3
junk
Instr3
jmp Instr4
junk
End:
```

```
jmp Start
Instr2
jmp Instr3
junk
Instr3
jmp Instr4
junk
Instr5
jmp End
Start:
Instr1
jmp Instr2
Instr4
jmp Instr5
End:
```

```
jmp Start
Instr3
Instr4
jmp Instr5
junk
Instr5
jmp End
Start:
Instr1
jmp Instr2
junk
Instr2
jmp Instr3
junk
End:
```

Instruction Permutation Detection

- Standard AV software uses an emulator to detect the effect of the code, rather than trying to statically analyze it
- “Detection via Normalization”
  - use existing compiler transformations to remove the “de-optimizations”
    - e.g., 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