CSE509 Computer System Security

Stony Brook University

2023-03-28

Reverse Engineering

Michalis Polychronakis

Stony Brook University

Anti-reverse Engineering

Create software that is difficult to reverse engineer

- Intellectual property protection (copy protection, DRM)
- Bytecode-based languages (Java, .NET) facilitate decompilation ➔ code obfuscators
- Evade malicious code analysis/detection (flip side: the mere presence of heavy obfuscation may raise suspicion)

It is never possible to entirely prevent reversing (!)

- Just obstruct reverse engineers by making the process very slow and tiring

The more effective an anti-reversing technique, the higher its cost

- CPU usage, code size/complexity, reliability, …

Main Anti-reverse Engineering Approaches

Symbolic information stripping

- Compiled programs: remove symbols
- Bytecode programs: rename symbols into meaningless values

Code obfuscation

- Change/complicate program code/layout/data

Anti-debugging and anti-analysis code

- Alter behavior when being debugged/analyzed

Runtime packing/encryption

Code virtualization

Anti-debugging Techniques

Mostly platform/debugger specific tricks

- Very challenging to cover all possible debuggers/analyzers

OS-specific APIs

- `IsDebuggerPresent`, `SystemKernelDebuggerInformation`, …

Set up own interrupt handler

- For `int 3`, `int 1`, or trap flag

Runtime code checksums

- Self-detect any code modifications made by debuggers or by manual code patching

Code Disassembly

Linear sweep (e.g., objdump)

- Decode everything as instructions, starting from the first byte
- Advantage: simplicity ➔ just linear instruction decoding
- Disadvantage: accuracy ➔ data may be misinterpreted as code

Recursive traversal (e.g., IDA Pro)

- Take into account the control flow of the code
- Linear sweep until a jump or branch instruction is encountered ➔ recursively continue at all targets
- Indirect branches cannot be followed (target is known only at run time)

Anti-disassembly Techniques (Code Obfuscation)

Intermix code with data

- Also: dead/unreachable code

Opaque predicates

- Conditional branches that in practice are unconditional

Jump in the middle of instructions

- Also: unaligned instructions

Function inlining

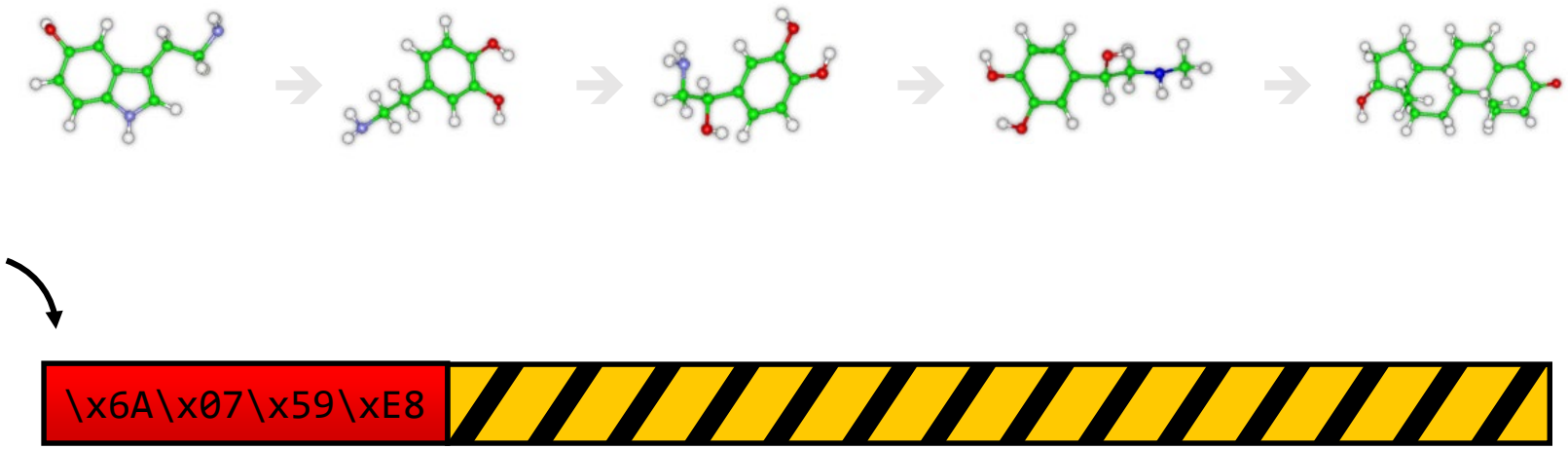
- Also the opposite: splitting linear code into multiple functions

Data transformation

- Encode or encrypt data, insert bogus data, restructure arrays, …

Many more: interleaved code, asynchronous code, self-modifying code, …

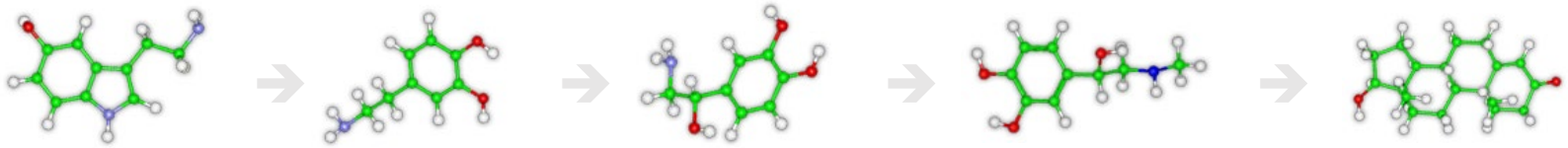
Polymorphism

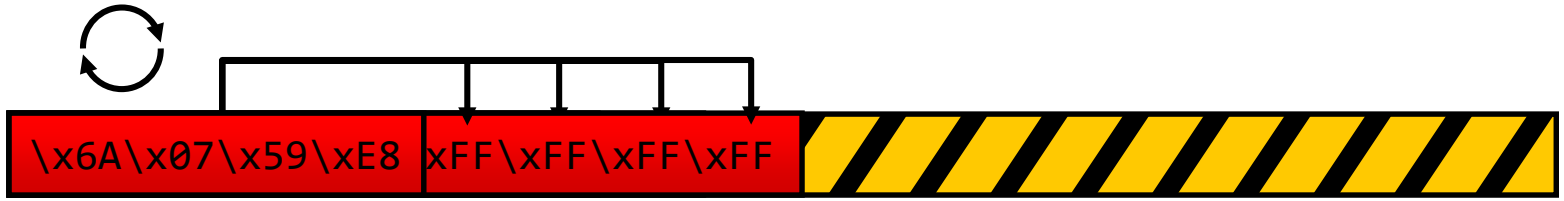


Use case: shellcode “packing”

- IDS evasion
- Avoidance of restricted bytes in the attack vector
- ASCII/Alphanumeric shellcode

Polymorphism

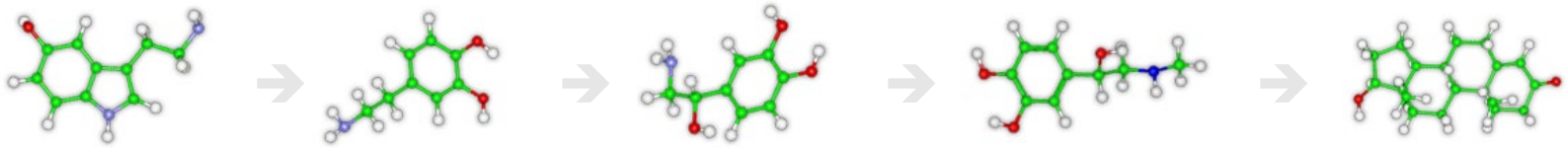


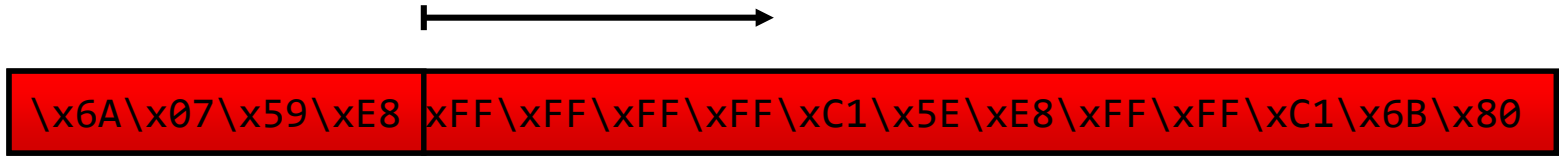


Use case: shellcode “packing”

- IDS evasion
- Avoidance of restricted bytes in the attack vector
- ASCII/Alphanumeric shellcode

Polymorphism





Use case: shellcode “packing”

- IDS evasion
- Avoidance of restricted bytes in the attack vector
- ASCII/Alphanumeric shellcode

Code Obfuscation (Metamorphism)

NOP interspersion

```
inc ecx
dec ecx
```

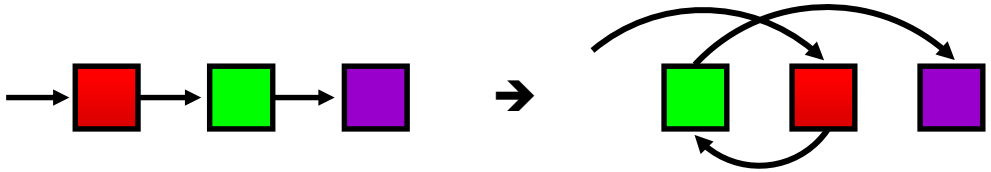
Instruction substitution

```
mov eax,0xF3
```

➔

```
push 0xF3
pop eax
```

Block transposition



Register reassignment

```
sed -i 's/eax/ebx/g'
```

Dead code insertion

All techniques can be combined!

Example: Static Analysis Resistant Shellcode

What does this code do?

```
\x6A\x0F\x59\xE8\xFF\xFF\xFF\xFF\xC1\x5E\x80\x46
\x0A\xE0\x30\x4C\x0E\x0B\x02\xFA...
```

Linear Disassembly

```
00  6A7F       push byte 0x7f
02  59         pop ecx
03  E8FFFFFFFF call 0x7
08  C15E8046   rcr [esi-0x80],0x46
0C  0AE0       or ah,al
0E  304C0E0B   xor [esi+ecx+0xb],cl
12  02FA       add bh,dl
14
... <encrypted shellcode>
93
```

Linear Disassembly

```
00  6A7F        push byte 0x7f
02  59          pop ecx
03  E8FFFFFFFF  call 0x7
08  C15E8046    rcr [esi-0x80],0x46
0C  0AE0        or ah,al
0E  304C0E0B    xor [esi+ecx+0xb],cl
12  02FA        add bh,dl
14
... <encrypted shellcode>
93
```

Jumps to the middle of itself

Linear Disassembly

```
00  6A7F       push byte 0x7f
02  59         pop ecx
03  E8FFFFFFFF call 0x7
08  C15E8046   rcr [esi-0x80],0x46
0C  0AE0       or ah,al
0E  304C0E0B   xor [esi+ecx+0xb],cl
12  02FA       add bh,dl
14
... <encrypted shellcode>
93
```

Recursive Traversal Disassembly

```
00  6A7F       push byte 0x7f
02  59         pop ecx
03  E8FFFFFFFF call 0x7
07  FFC1       inc ecx
09  5E         pop esi
0A  80460AE0   add [esi+0xa],0xe0
0E  304C0E0B   xor [esi+ecx+0xb],cl
12  02FA       add bh,dl
14
... <encrypted shellcode>
93
```

Linear Disassembly

```
00  6A7F       push byte 0x7f
02  59         pop ecx
03  E8FFFFFFFF call 0x7
08  C15E8046   rcr [esi-0x80],0x46
0C  0AE0       or ah,al
0E  304C0E0B   xor [esi+ecx+0xb],cl
12  02FA       add bh,dl
14
... <encrypted shellcode>
93
```

Recursive Traversal Disassembly

```
00  6A7F       push byte 0x7f
02  59         pop ecx
03  E8FFFFFFFF call 0x7
07  FFC1       inc ecx
09  5E         pop esi
0A  80460AE0   add [esi+0xa],0xe0
0E  304C0E0B   xor [esi+ecx+0xb],cl
12  02FA       add bh,dl
14
... <encrypted shellcode>
93
```

Linear Disassembly

```
00  6A7F        push byte 0x7f
02  59          pop ecx
03  E8FFFFFFFF  call 0x7
08  C15E804     [illegible]
0C  0AE0        or ah,al
0E  304C0E0B    xor [esi+ecx+0xb],cl
12  02FA        add bh,dl
14
... <encrypted shellcode>
93
```

Recursive Traversal Disassembly

```
00  6A7F        push byte 0x7f
02  59          pop ecx
03  E8FFFFFFFF  call 0x7
[illegible]     [illegible]
09  5E          pop esi
0A  80460AE0    add [esi+0xa],0xe0
[illegible]     [illegible] [esi+ecx+0xb],cl
12  02FA        add bh,dl
14
... <encrypted shellcode>
93
```

still not the actual code

that will be executed!

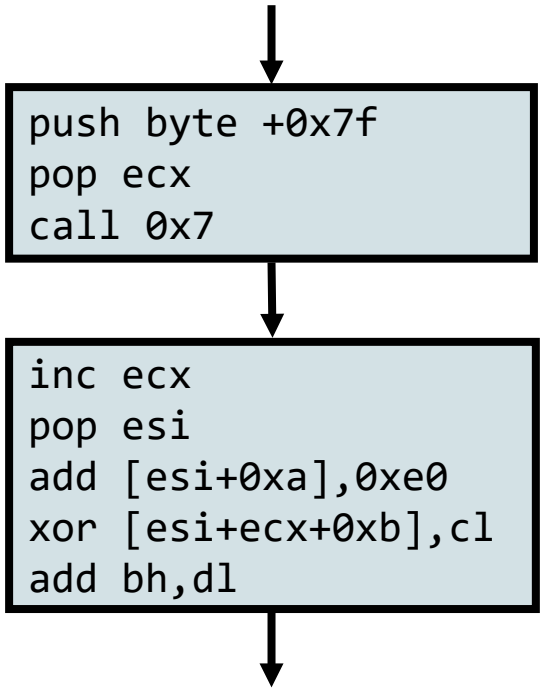
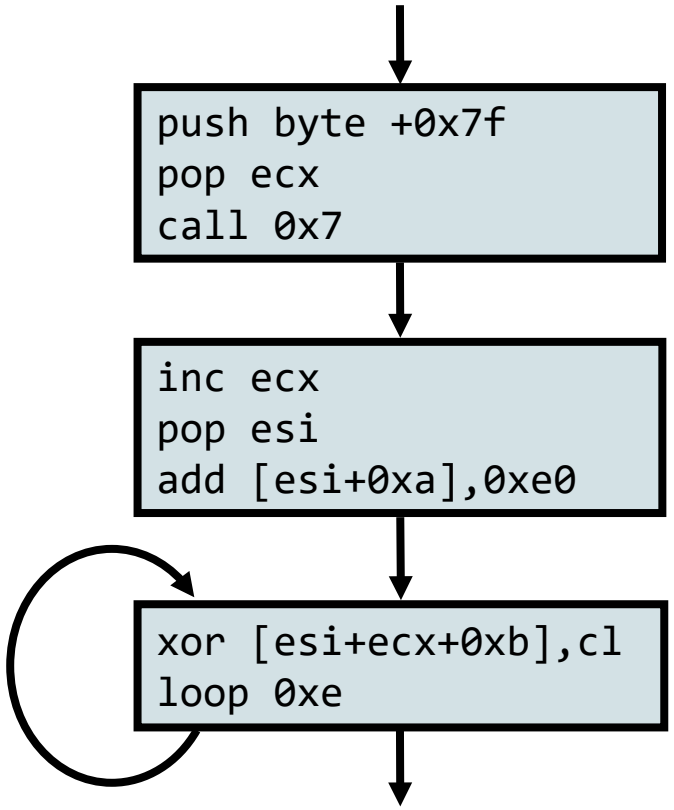
Recursive Traversal Disassembly			Real Code Execution
00	6A7F	push byte 0x7f	
02	59	pop ecx	ecx = 0x7F
03	E8FFFFFFFF	call 0x7	push 0x8
07	FFC1	inc ecx	ecx = 0x80
09	5E	pop esi	esi = 0x8
0a	80460AE0	add [esi+0xa],0xe0	ADD [12] 0xE0
0e	304C0E0B	xor [esi+ecx+0xb],cl	
12	02FA	add bh,dl	
14			
...	<encrypted shellcode>		
93			

Recursive Traversal Disassembly			Real Code Execution
00	6A7F	push byte 0x7f	
02	59	pop ecx	ecx = 0x7F
03	E8FFFFFFFF	call 0x7	push 0x8
07	FFC1	inc ecx	ecx = 0x80
09	5E	pop esi	esi = 0x8
0a	80460AE0	add [esi+0xa],0xe0	ADD [12] 0xE0
0e	304C0E0B	xor [esi+ecx+0xb],cl	
12	**E2FA**	**loop 0xe**	
14			
...	<encrypted shellcode>		
93			

self-modification

Recursive Traversal Disassembly			Real Code Execution
00	6A7F	push byte 0x7f	
02	59	pop ecx	ecx = 0x7F
03	E8FFFFFFFF	call 0x7	push 0x8
07	FFC1	inc ecx	ecx = 0x80
09	5E	pop esi	esi = 0x8
0a	80460AE0	add [esi+0xa],0xe0	ADD [12] 0xE0
0e	304C0E0B	xor [esi+ecx+0xb],cl	XOR [93] 0x80
12	**E2FA**	**loop 0xe**	ecx = 0x7F
14			XOR [92] 0x7F
...	<encrypted	shellcode>	ecx = 0x7E
93			XOR [91] 0x7E
			...

Static Analysis



Actual Execution