AUTOMATIC GENERATION OF COMPACT PRINTABLE SHELLCODES FOR x86

WOOT ’20

DHRUMIL PATEL
ADITYA BASU
ANISH MATHURIA

AUGUST 11, 2020
Introduction
- Currently used Algorithms
- Motivation
- Printable Shellcode Compiler (psc)
- Results
- Conclusion
Printable Shellcodes

- Defensive filters strip all the printable characters from input.
- This ruins most injection attacks.

Attacker’s Goal is to generate code that consists only of:
- 0123456789
- ABCDEFGHIJKLMNOPQRSTUVWXYZ
- abcdefghijklmnopqrstuvwxyz
- !"#$%&'()*+,-./;<>?@[\]^_`{|}~
Currently Used Algorithms

Riley Eller Algorithm

“Any dword (4 bytes) can be derived from two or three SUB instructions whose operands are printable bytes”

Available as a Metasploit plugin

Ex. 0x89e3cd80 bytes from execv /bin/sh shellcode

```
# Constants are printable (0x21-0x7E)
sub $0x256d6d2d, %eax
sub $0x256d6d25, %eax
sub $0x34574225, %eax
push %eax
```

Each byte is encoded with ≥ 2.5 bytes

Source to Source Conversion

Geczi and Ivanyi replace all non-printable instructions with a sequence of printable instruction(s)

38 byte shellcode → 9837 bytes printable shellcode

Not publicly available
We Need Compact Shellcodes

- Transforming to printable $\Rightarrow$ increases shellcode size
- Size restrictions on input

Example
Buffer overflow exploits are limited by the buffer size.
PRINTABLE SHELLCODE COMPILER

OVERVIEW
ENCODING SCHEME
RUNTIME DECODER
TESTING
psc: PRINTABLE SHELLCODE COMPILER

- Arbitrary shellcode → Printable shellcode
- Special encoding scheme
- Hand-crafted decoder that is printable

Diagram:

- Decoder (size = 146 bytes)
- Encoded shellcode
  - Custom XOR Patcher
  - Decoder loop (size = 73 bytes)
  - Encoded payload

Printable shellcode
**psc Encoding Scheme**

Original Byte #1

| b₀ | b₁ | b₂ | b₃ | b₄ | b₅ | b₆ | b₇ |

Original Byte #2

| b₈ | b₉ | b₁₀| b₁₁| b₁₂| b₁₃| b₁₄| b₁₅|

Encoded B1 = 0x3F +

| 0  | 0  | 0  | 0  | b₀ | b₁ | b₂ | b₃ |

Encoded B2 = 0x3F +

| 0  | 0  | b₄ | b₅ | b₆ | b₇ | b₈ | b₉ |

Encoded B3 = 0x3F +

| 0  | 0  | b₁₀| b₁₁| b₁₂| b₁₃| b₁₄| b₁₅|

Range of Encoded Bytes

\((0x3F, 0x7E)\)
**psc Runtime Decoding**

**Initializer**

- ECX: Read Pointer
- EDX: Write Pointer

**Loop**

Let,

- B1 ← [ECX]
- B2 ← [ECX + 1]
- B3 ← [ECX + 2]

**Decision**

- B1 = 0x26
  - NO
  - YES

**Sub-Loop**

- Recover first byte,
  - R1 ← (B1 << 4) + (B2 & 0x3F) >> 2

- Recover second byte,
  - R2 ← (B2 << 6) + (B3 & 0x3F)

- Write R1 to [EDX]
- Write R2 to [EDX+1]

**Recovered Shellcode**

- EDX ← EDX + 2
- ECX ← ECX + 3

**Final Decision**

- 3 bytes encoded payload → 2 bytes recovered shellcode

- Read ‘&’ (=0x26) → jump to recovered shellcode
Signal handler checks

RECOVERED SHELLCODE == ORIGINAL SHELLCODE
Shellcode to spawn shell on 4444/TCP

```
x31\xc0\x31\xdb\x53\x43\x53\x6a\x80\xe1\xb0\x66\xcd\x80\x97\x31\xc0\x43\x50\x66\x68\x11\x5c\x66\x53\x89\xe1\x6a\x10\x51\x57\x89\xe1\xb0\x66\xcd\x80\x50\x57\x89\xe1\xb0\x66\x83\xc3\x02\xcd\x80\x50\x57\x89\xe1\x04\x66\x43\xcd\x80\x93\x31\xc0\x31\xc9\xb1\x02\xb0\x3f\xcd\x80\x49\x79\xf9\x50\x66\x68\x73\x68\x68\x2f\x62\x61\x68\x2f\x62\x69\x6e\xe8\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80
```

Find at https://github.com/dhrumil29699/Printable-Encoder/
RESULTS

ENCODING PERFORMANCE
TOTAL OUTPUT SIZE
**RESULTS: ONLY ENCODED SHELLCODE**

**Encoding Performance**

- **Shell on 8080/TCP over SSL**
- **HTTP Server on 8800/TCP**
- **Download file & execute**
- **Shell on 4444/TCP**
- **Copy /etc/passwd**
- **Add root user**
- **Execve /bin/sh**

`psc` encoding is more compact than *Riley Eller* algorithm.
For large shellcodes, $psc$ beats the Riley Eller algorithm.
psc vs ALPHA3

- psc encoding outperforms ALPHA3.
  
  ALPHA3 changes 1 byte → 2 bytes  
  psc changes 1 byte → 1.5 bytes

- However, the compact encoding makes our decoder complicated.

- psc beats ALPHA3 for larger shellcodes (size > 236 bytes).
We present a new encoding algorithm that uses looped decoding to reduce the size of the auto-generated printable shellcodes.

We produce about 40% – 50% smaller printable shellcodes as compared to the Riley Eller algorithm.

Future Plan
Add support for x86_64 shellcodes
REFERENCES

RILEY ELLER
BYPASSING MSB DATA FILTERS FOR
BUFFER OVERFLOW EXPLOITS ON INTEL PLATFORMS

ZSOLT GÉCZI AND PETER IVÁNYI (2018)
AUTOMATIC TRANSLATION OF ASSEMBLY SHELLCODES
TO PRINTABLE BYTE CODES

B.J. WEVER
ALPHA³
THANKS!

Dhruvil Patel
Devops at Acko Technology and Services Private Ltd.
2016@1228@daiict.ac.in

Aditya Basu
PhD Student at Penn State
aditya.basu@psu.edu

Anish Mathuria
Professor at DA-IICT
anish_mathuria@daiict.ac.in