

"Guardians of the Secrets: Analyzing Cryptographic Algorithms in Malicious Code"

Muhammad Ahmad

[Kill_The_Malware]

RC4 Unveiled : The Three Stages of Secrets - KSA, PRGA, and XOR Operations"

- **Rc4 have 3 stages**
 - KSA [key Scheduling Algorithm]
 - PRGA [Pseudo Random Generation Algo]
 - XOR operations
- **KSA Stage:**
 - During initialization, a list of values from 0 to 255 is created. These values are rearranged by swapping them based on calculations involving two indices, S[j] and S[i].
- **PRGA [Pseudo Random Generation Algo]**
 - Generates key stream by rearranging values in the list.
 - Produces bytes, limited up to 256.
 - Using modulo % with 256 for simplicity.
 - 3 iterations using 256, 1 swapping step.

```
RC4(const std::vector<unsigned char>& key) : S(256), i(0), j(0) {  
    for (int i = 0; i < 256; ++i) {  
        S[i] = i;  
    }  
  
    int j = 0;  
    for (int i = 0; i < 256; ++i) {  
        j = (j + S[i] + key[i % key.size()]) % 256;  
        std::swap(S[i], S[j]);  
    }  
}  
  
unsigned char generateByte() {  
    i = (i + 1) % 256;  
    j = (j + S[i]) % 256;  
    std::swap(S[i], S[j]);  
    return S[(S[i] + S[j]) % 256];  
}  
  
for i in range(N):  
    encrypted_byte = plaintext[i] XOR keystream[i]  
    ciphertext.append(encrypted_byte)
```

KSA

PRGA

XOR

Rc4 in Dharma malware

● Rc4 code in C

```
def rc4_encrypt(data, key):
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]

    i = 0
    j = 0
    encrypted = bytearray()
    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        keystream_byte = S[(S[i] + S[j]) % 256]
        encrypted.append(byte ^ keystream_byte)

    return bytes(encrypted)

key = b'SecretKey'
plaintext = b'This is a secret message.'

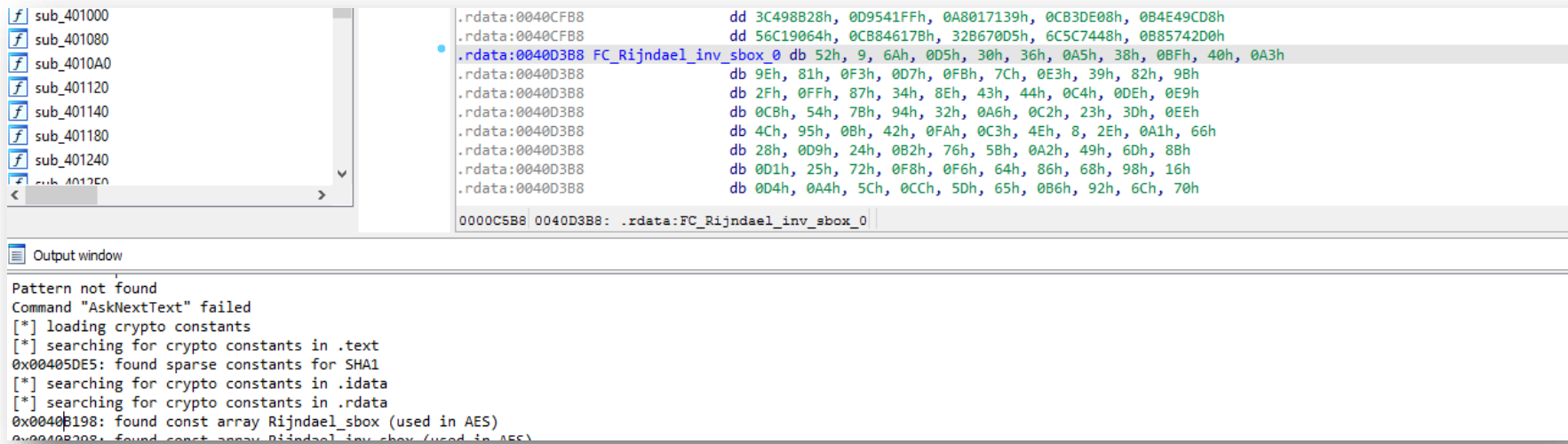
encrypted = rc4_encrypt(plaintext, key)
```

Dharma Code analysis

```
sub_401000(v17, 256);
v3 = 0;
v4 = a1;
v5 = v19;
v7 = v4 - v6;
do
{
    v8 = &v17[v3];
    v9 = *(_BYTE *)((v3 & 0x1F) + v5);
    v8[v7] = v3++;
    *v8 = v9;
}
while ( v3 < 256 );
v10 = v18;
v11 = (unsigned __int8 *)v18;
v12 = &v17[-v18];
do
{
    v13 = *v11;
    v14 = (v21 + (char)v11[(DWORD)v12] + *v11) % 256;
    result = *(_BYTE *) (v14 + v10);
    *v11++ = result;
    v16 = v20-- == 1;
    v21 = v14;
    // BYTE $V16 = 0;
}
```

AES Revealed: Unmasking AES with Find-Crypt

- Discovering the AES algorithm in malware samples is made easier by examining lookup tables like S-Boxes or T-Tables, depending on the implementation of AES.
- To expedite the search process, tools such as Find-Crypt can be used. [Findcrypt](#) is an IDA Python plugin designed for recognizing algorithm constants.
- In the specific case mentioned (REvil), Find-Crypt not only identified constants related to the Salsa20 encryption algorithm but also recognized AES tables, which are indicative of AES encryption being used within the malware.
- string Rijndael_inv_sbox which are associated with the AES encryption algorithm checkout for AES Source Code <https://android.googlesource.com/platform/external/openssh/+/idea133/rijndael.c>



```
sub_401000
sub_401080
sub_4010A0
sub_401120
sub_401140
sub_401180
sub_401240
sub_401260

.rdata:0040CFB8 dd 3C498828h, 0D9541FFh, 0A8017139h, 0CB3DE08h, 0B4E49CD8h
.rdata:0040CFB8 dd 56C19064h, 0CB84617Bh, 32B670D5h, 6C5C7448h, 0B85742D0h
.rdata:0040D3B8 FC_Rijndael_inv_sbox_0 db 52h, 9, 6Ah, 0D5h, 30h, 36h, 0A5h, 38h, 0BFh, 40h, 0A3h
.rdata:0040D3B8 db 9Eh, 81h, 0F3h, 0D7h, 0FBh, 7Ch, 0E3h, 39h, 82h, 98h
.rdata:0040D3B8 db 2Fh, 0FFh, 87h, 34h, 8Eh, 43h, 44h, 0C4h, 0DEh, 0E9h
.rdata:0040D3B8 db 0CBh, 54h, 78h, 94h, 32h, 0A6h, 0C2h, 23h, 3Dh, 0EEh
.rdata:0040D3B8 db 4Ch, 95h, 08h, 42h, 0FAh, 0C3h, 4Eh, 8, 2Eh, 0A1h, 66h
.rdata:0040D3B8 db 28h, 0D9h, 24h, 0B2h, 76h, 58h, 0A2h, 49h, 6Dh, 88h
.rdata:0040D3B8 db 0D1h, 25h, 72h, 0F8h, 0F6h, 64h, 86h, 68h, 98h, 16h
.rdata:0040D3B8 db 0D4h, 0A4h, 5Ch, 0CCh, 5Dh, 65h, 0B6h, 92h, 6Ch, 70h

0000C5B8 0040D3B8: .rdata:FC_Rijndael_inv_sbox_0

Output window
Pattern not found
Command "AskNextText" failed
[*] loading crypto constants
[*] searching for crypto constants in .text
0x00405DE5: found sparse constants for SHA1
[*] searching for crypto constants in .idata
[*] searching for crypto constants in .rdata
0x0040B198: found const array Rijndael_sbox (used in AES)
0x0040B208: found const array Rijndael_inv_sbox (used in AES)
```

The Hidden Secrets of Blowfish Implementation

- Discovering the Similarities include the presence of two loops, each iterating 18 times, which are employed for initializing the P-Array.
- Additionally, there is a loop that iterates four times, with an internal loop that runs 256 times, serving the purpose of initializing the S-Boxes.

Assembly Code	BlowFish.c
<pre>memmove(v16 + 4, &unk_5605C0, 0x48u); memmove(v16 + 22, &unk_560608, 0x1000u); v11 = v17; v13 = 0; v10 = 0; for (i = 0; i < 0x12; ++i) { v13 = 0; v7 = 4; while (v7--) { v13 = (v13 << 8) (unsigned __int8)*v11++; if (++v10 == Size) { v10 = 0; v11 = v17; } } v16[i + 4] ^= v13; } v8 = 0; v9 = 0; for (i = 0; i < 0x12; ++i) { sub_422610(&v8); v16[i + 4] = v8; v16[i + 4] = v9; } for (j = 0; j < 4; ++j) { for (k = 0; k < 256; ++k) { sub_422610(&v8); } }</pre>	<pre>memcpy(keystruct->p,p_perm,sizeof(WORD) * 18); memcpy(keystruct->s,s_perm,sizeof(WORD) * 1024); // Combine the key with the P box. Assume key is standard 448 bits (56 bytes) or less. for (idx = 0, idx2 = 0; idx < 18; ++idx, idx2 += 4) keystruct->p[idx] ^= (user_key[idx2 % len] << 24) (user_key[(idx2+1) % len] << 16) (user_key[(idx2+2) % len] << 8) (user_key[(idx2+3) % len]); // Re-calculate the P box. memset(block, 0, 8); for (idx = 0; idx < 18; idx += 2) { blowfish_encrypt(block,block,keystruct); keystruct->p[idx] = (block[0] << 24) (block[1] << 16) (block[2] << 8) block[3]; keystruct->p[idx+1] = (block[4] << 24) (block[5] << 16) (block[6] << 8) block[7]; } // Recalculate the S-boxes. for (idx = 0; idx < 4; ++idx) { for (idx2 = 0; idx2 < 256; idx2 += 2) { blowfish_encrypt(block,block,keystruct); keystruct->s[idx][idx2] = (block[0] << 24) (block[1] << 16) (block[2] << 8) block[3]; keystruct->s[idx][idx2+1] = (block[4] << 24) (block[5] << 16) (block[6] << 8) block[7]; } }</pre>