[QuickNote] Decrypting the C2 configuration of Warzone RAT

kienmanowar.wordpress.com/2023/03/25/quicknote-decrypting-the-c2-configuration-of-warzone-rat/

March 25, 2023

1. Introduction

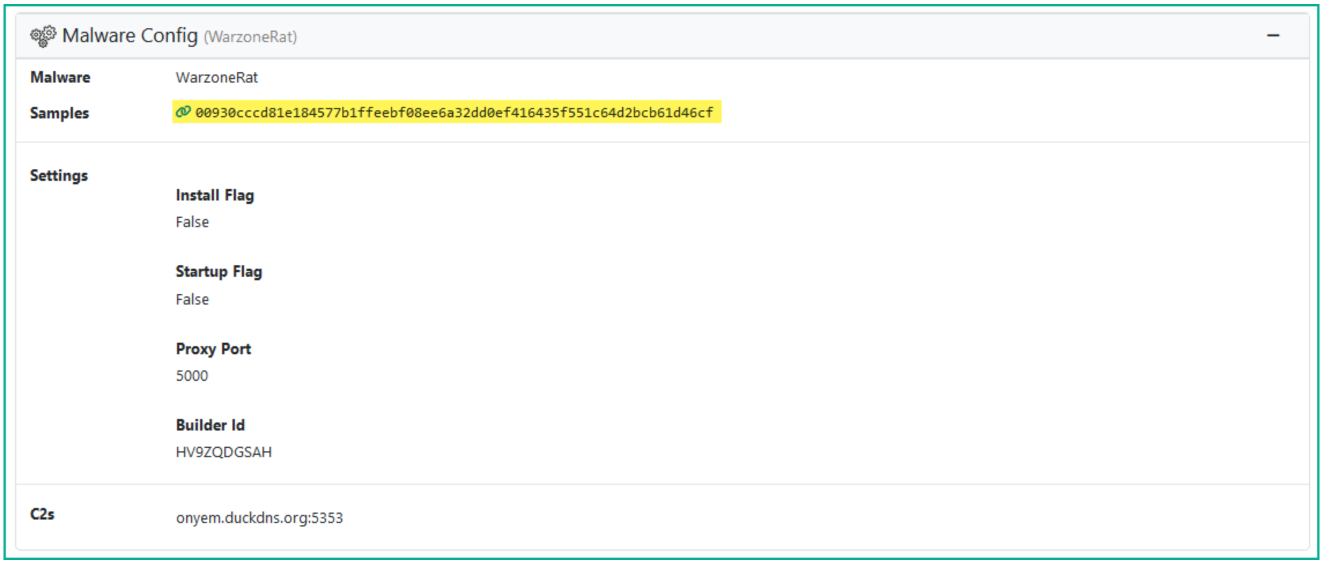
Warzone RAT is a type of malware that is capable of infiltrating a victim's computer and giving attackers remote access and control over the system. The malware has gained notoriety for its advanced capabilities and ability to evade detection, making it a serious threat to computer security.

Warzone RAT is typically spread through phishing emails or other social engineering techniques, where attackers trick victims into downloading and installing the malware on their systems. Once the malware is installed, it can perform a variety of malicious actions, including stealing passwords, taking screenshots, and logging keystrokes. It can also download and execute additional malware, giving attackers even more control over the victim's system.

One of the key features of Warzone RAT is its ability to encrypt its configuration data, making it difficult for security experts to analyze and understand how the malware operates. *Currently, there are two variants of the malware in circulation, each using a different method to decode its configuration. The first variant uses standard RC4 encryption, while the second variant uses a modified version of RC4. This modification makes it even more challenging to decrypt and analyze the malware's configuration data.*

2. Analysis

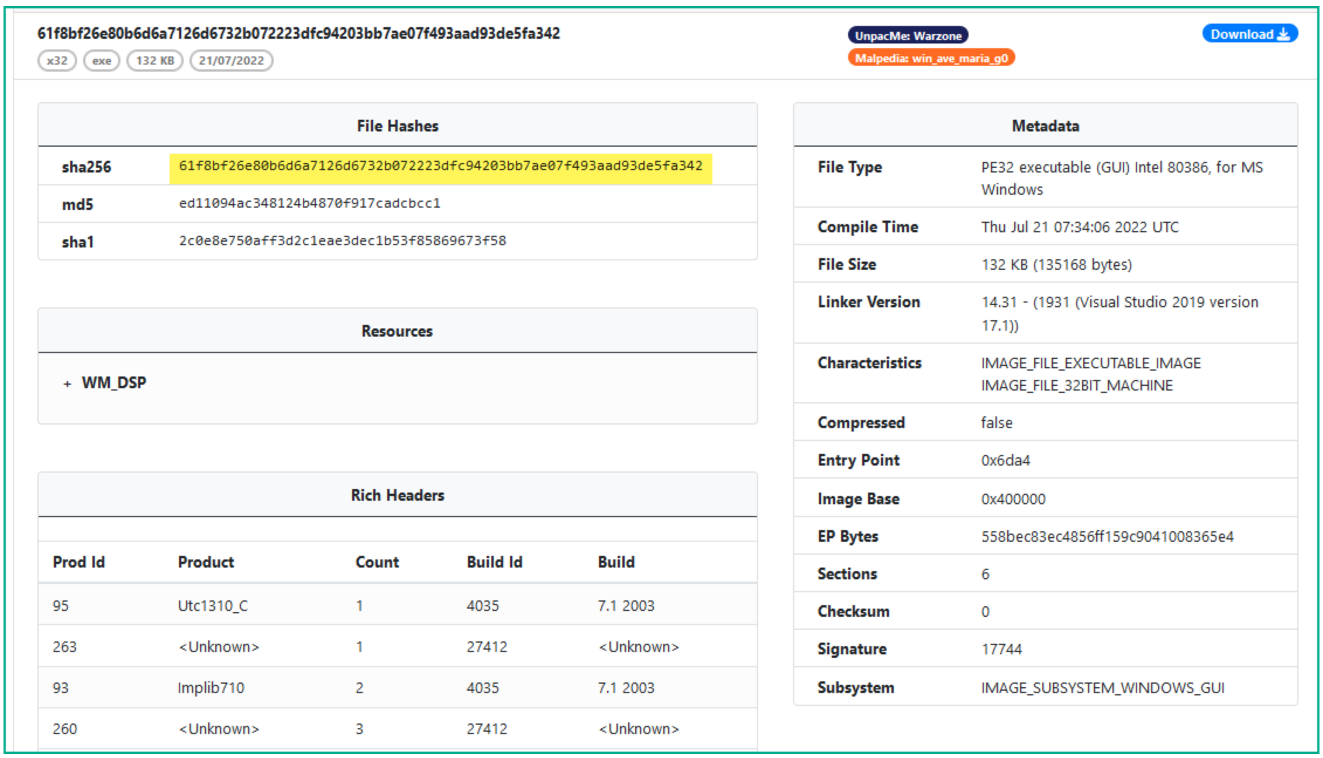
Sample1: 00930cccd81e184577b1ffeebf08ee6a32dd0ef416435f551c64d2bcb61d46cf (use standard RC4)



Malware Config (WarzoneRat)

Malware	WarzoneRat
Samples	00930cccd81e184577b1ffeebf08ee6a32dd0ef416435f551c64d2bcb61d46cf
Settings	**Install Flag** False **Startup Flag** False **Proxy Port** 5000 **Builder Id** HV9ZQDGSAH
C2s	onyem.duckdns.org:5353

Sample2: 61f8bf26e80b6d6a7126d6732b072223dfc94203bb7ae07f493aad93de5fa342 (use modified RC4)



61f8bf26e80b6d6a7126d6732b072223dfc94203bb7ae07f493aad93de5fa342

x32 exe 132 KB 21/07/2022

UnpacMe: Warzone

Malpedia: win_ave_maria_g0

Download

File Hashes

sha256	61f8bf26e80b6d6a7126d6732b072223dfc94203bb7ae07f493aad93de5fa342
md5	ed11094ac348124b4870f917cadcbcc1
sha1	2c0e8e750aff3d2c1eae3dec1b53f85869673f58

Resources

+ WM_DSP

Rich Headers

Prod Id	Product	Count	Build Id	Build
95	Utc1310_C	1	4035	7.1 2003
263	<Unknown>	1	27412	<Unknown>
93	Implib710	2	4035	7.1 2003
260	<Unknown>	3	27412	<Unknown>

Metadata

File Type	PE32 executable (GUI) Intel 80386, for MS Windows
Compile Time	Thu Jul 21 07:34:06 2022 UTC
File Size	132 KB (135168 bytes)
Linker Version	14.31 - (1931 (Visual Studio 2019 version 17.1))
Characteristics	IMAGE_FILE_EXECUTABLE_IMAGE IMAGE_FILE_32BIT_MACHINE
Compressed	false
Entry Point	0x6da4
Image Base	0x400000
EP Bytes	558bec83ec4856ff159c9041008365e4
Sections	6
Checksum	0
Signature	17744
Subsystem	IMAGE_SUBSYSTEM_WINDOWS_GUI

In Warzone RAT, the configuration info is stored in the `.bss` PE section of the malware's code. The `.bss` section is typically used for storing uninitialized data. The format of the configuration is as follows: **`[Key length] [RC4 key] [Encrypted data]`**. Below is an illustration of the configuration stored in the `.bss` section in both samples.

The steps to perform the process of retrieving information and copying data from the `.bss` section to memory are the same in both samples. The pseudo-code is shown below:

```
BOOL __usercall wzr_decrypt_config_stored_at_bss@<eax>(struct_this_20 *arg_struct_cfg@<ecx>, int arg_zero_value@<ebx>)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Sleep(0x1F4u);
  wzr_init_struct_16(&struct_payload_info);
  wzr_payload_base_addr = wzr_get_base_addr_of_curr_pe_file();
  wzr_copy_pe_headers_and_sections_info(&struct_payload_info, wzr_payload_base_addr);
  var_str_bss = wzr_strcpy(&var_temp_data.cbData, arg_zero_value, ".bss");
  wzr_grab_info_of_bss_section(&struct_payload_info, &source_data, var_str_bss);
  wzr_VirtualFree(var_temp_data.cbData);
  wzr_clone_data(&var_temp_data, &source_data.section_address);
  wzr_clone_data2(&arg_struct_cfg->ptr_bss_content, &var_temp_data);
  wzr_free_heap_and_reset_pointer(&var_temp_data.pbData);                    Sample 1
  wzr_decrypt_config(arg_struct_cfg, &v23);
```

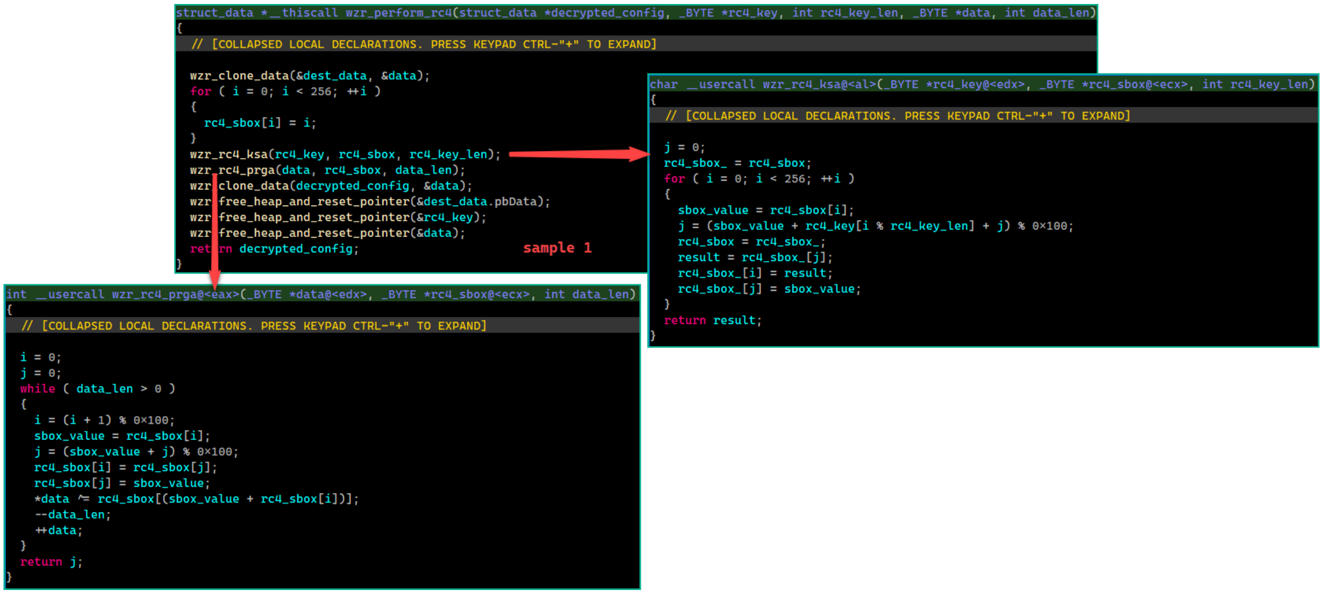
```
BOOL __usercall wzr_decrypt_config_stored_at_bss@<eax>(wzr_struct_21 *arg_struct_cfg@<ecx>, int arg_zero_value@<ebx>)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Sleep(0x1F4u);
  wzr_init_struct_16(&struct_payload_info);
  wzr_payload_base_addr = wzr_get_base_addr_of_curr_pe_file();
  wzr_copy_pe_headers_and_sections_info(&struct_payload_info, wzr_payload_base_addr);
  var_str_bss = wzr_strcpy(&var_temp_data.cbData, arg_zero_value, ".bss");
  wzr_grab_info_of_bss_section(&struct_payload_info, &foundSectionInfo, var_str_bss);
  wzr_VirtualFree(var_temp_data.cbData);
  wzr_clone_data(&var_temp_data, &foundSectionInfo.section_address);
  wzr_clone_data2(&arg_struct_cfg->ptr_bss_content, &var_temp_data);           Sample 2
  wzr_free_heap_and_reset_pointer(&var_temp_data.pbData);
  wzr_decrypt_config(arg_struct_cfg, &decrypted_config);
```

The pseudo code in function `wzr_decrypt_config` in both samples is the same, which involves extracting the RC4 Key and Encrypted data, and then using RC4 to decrypt the configuration. The difference lies in function `wzr_perform_rc4`.

```
int __thiscall wzr_decrypt_config(struct_this_20 *arg_struct_this_20, int decrypted_config)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  wzr_copy_from_offset(
    &arg_struct_this_20->ptr_bss_content.pbData,
    &var_encrypted_config,
    *arg_struct_this_20->ptr_bss_content.pbData + 4,
    arg_struct_this_20->ptr_bss_content.cbData - *arg_struct_this_20->ptr_bss_content.pbData - 4);
  var_encryped_data.cbData = v3;
  var_encryped_data.pbData = v3;
  wzr_clone_data(&var_encryped_data, &var_encrypted_config);
  var_rc4_key.cbData = v4;
  var_rc4_key.pbData = v4;
  wzr_copy_data_0(arg_struct_this_20, &var_rc4_key);
  wzr_perform_rc4(decrypted_config, var_rc4_key.pbData, var_rc4_key.cbData, var_encryped_data.pbData, var_encryped_data.cbData);
  wzr_free_heap_and_reset_pointer(&var_encrypted_config.pbData);
  return decrypted_config;                                                  Sample 1
}
```

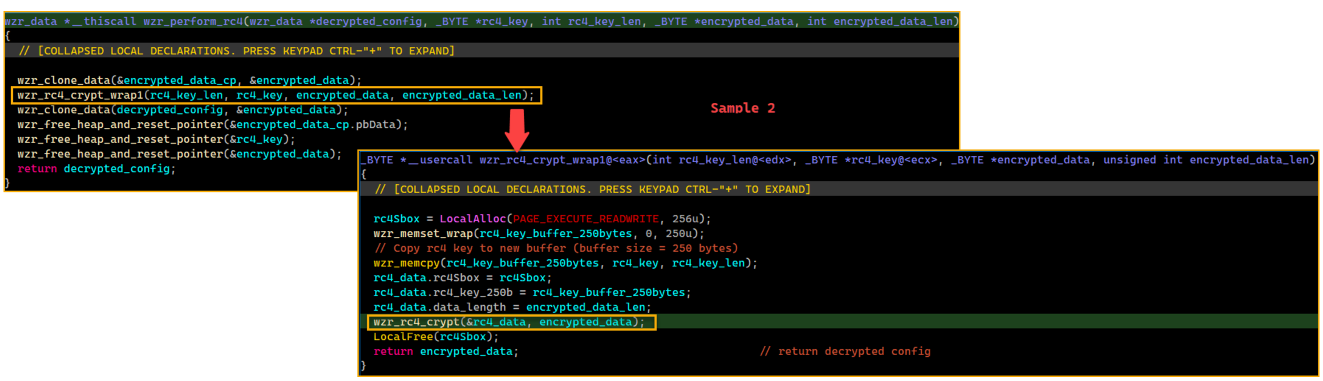
```
wzr_data *__thiscall wzr_decrypt_config(wzr_struct_21 *this, wzr_data *decrypted_config)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  wzr_clone_data3(
    &this->ptr_bss_content.pbData,
    &var_encrypted_config,
    *this->ptr_bss_content.pbData + 4,
    this->ptr_bss_content.cbData - *this->ptr_bss_content.pbData - 4);
  var_encrypted_config_cp.cbData = v3;
  var_encrypted_config_cp.pbData = v3;
  wzr_clone_data(&var_encrypted_config_cp, &var_encrypted_config);
  var_rc4_key.cbData = v4;
  var_rc4_key.pbData = v4;
  wzr_clone_data4(this, &var_rc4_key);
  wzr_perform_rc4(decrypted_config, var_rc4_key.pbData, var_rc4_key.cbData, var_encrypted_config_cp.pbData, var_encrypted_config_cp.cbData);
  wzr_free_heap_and_reset_pointer(&var_encrypted_config.pbData);
  return decrypted_config;                                                  Sample 2
}
```

The function `wzr_perform_rc4` in sample 1 uses standard RC4 to decrypt the configuration. Its pseudocode is shown below:



Thus, we can easily use CyberChef to perform configuration decoding or write a Python script to automate for similar samples.

The pseudocode for function `wzr_perform_rc4` in sample 2 as shown below. Prior to decryption, it allocates an array of 250 bytes, filled with zero values. Then, it copies the extracted `rc4_key` into this array. Finally, it calls the `wzr_rc4_crypt` function, which uses the **modified RC4** algorithm to decrypt the configuration.



The complete pseudocode of the `wzr_rc4_crypt` function is as follows:

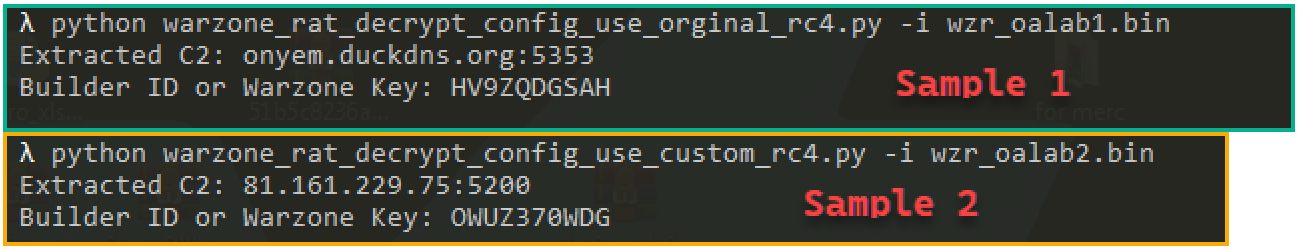
```
void __thiscall wzr_rc4_crypt(wzr_rc4_data *rc4_info, _BYTE *data)
{
  idx = 0;
  if ( rc4_info->rc4Sbox )
  {
    if ( rc4_info->rc4_key_250b )
    {
      rc4_info->counter2 = 0;
      LOBYTE(i) = 0;
      rc4_info->counter1 = 0;

      do
      {
        rc4_info->rc4Sbox[i] = rc4_info->counter1;
        i = rc4_info->counter1 + 1;
        rc4_info->counter1 = i;
      }
      while ( i < 256 );

      rc4_info->counter1 = 0;
      for ( i = 0; i < 256; rc4_info->counter1 = i )
      {
        rc4Sbox = rc4_info->rc4Sbox;
        rc4_info->counter2 += rc4Sbox[i] + rc4_info->rc4_key_250b[i % 250];
        rc4Sbox[i] ^= rc4Sbox[rc4_info->counter2];
        // swap values
        rc4_info->rc4Sbox[LOBYTE(rc4_info->counter2)] ^= rc4_info-
>rc4Sbox[LOBYTE(rc4_info->counter1)];
        rc4_info->rc4Sbox[LOBYTE(rc4_info->counter1)] ^= rc4_info-
>rc4Sbox[LOBYTE(rc4_info->counter2)];
        i = rc4_info->counter1 + 1;
      }
      rc4_info->counter1 = 0;
      rc4_info->counter2 = 0;
      // Decrypt data
      if ( rc4_info->data_length )
      {
        j = 0;
        do
        {
          rc4_info->counter1 = j + 1;
          rc4Sbox = rc4_info->rc4Sbox;
          k = (j + 1);
          rc4Sbox_value1 = rc4Sbox[k];
          rc4_info->counter2 += rc4Sbox_value1;
          rc4Sbox_value1_ = rc4Sbox_value1;
          rc4Sbox_value2 = rc4Sbox[rc4_info->counter2];
          rc4Sbox[k] = rc4Sbox_value2;
          rc4_info->rc4Sbox[LOBYTE(rc4_info->counter2)] = rc4Sbox_value1;
          rc4Sbox_ = rc4_info->rc4Sbox;
          data[idx] ^= rc4Sbox_[(rc4_info->counter2 + rc4Sbox_value2)] ^
(rc4Sbox_[(rc4Sbox_value2 + rc4Sbox_value1_)]
```

```
                                                                        +
rc4Sbox_[(rc4Sbox_[((0x20 * rc4_info->counter2) ^ (rc4_info->counter1 >> 3))]
                                                                                  +
rc4Sbox_[((0x20 * rc4_info->counter1) ^ (rc4_info->counter2 >> 3))]) ^ 0xAA]);
          j = ++rc4_info->counter1;
          ++idx;
        }
        while ( idx < rc4_info->data_length );
      }
    }
  }
}
```

With the pseudocode above, we can rewrite the decoding code in Python as follows. This is the code I wrote, and you can write it in your own way as long as it performs the task correctly.

```
# Refs: https://stackoverflow.com/questions/9433541/movsx-in-python
def SIGNEXT(x, b):
    m = (1 << (b -1))
    x = x & ((1 << b) -1)
    return ((x ^ m) - m)

# This routine is responsible for decrypting the stored C2.
def rc4_customized_decryptor(data, key):
    idx = 0
    counter1 = 0
    counter2 = 0

    # Initialize RC4 S-box
    rc4Sbox = list(range(256))

    # Modify RC4 S-box
    for i in range(256):
        counter2 += (rc4Sbox[i] + key[i%250])
        counter2 = counter2 & 0x000000FF
        rc4Sbox[i] ^= rc4Sbox[counter2]
        rc4Sbox[counter2 & 0xFF] ^= rc4Sbox[counter1 & 0xFF]
        rc4Sbox[counter1 & 0xFF] ^= rc4Sbox[counter2 & 0xFF]
        counter1 = i+1

    # Decrypt data
    counter1 = 0
    counter2 = 0
    j = 0
    decrypted = []
    while(idx < len(data)):
        counter1 = j + 1
        k = (j+1)
        rc4Sbox_value1 = rc4Sbox[k]
        counter2 += (SIGNEXT(rc4Sbox_value1, 8) & 0xFFFFFFFF)
        rc4Sbox_value1_ = (SIGNEXT(rc4Sbox_value1, 8) & 0xFFFFFFFF)
        rc4Sbox_value2 = rc4Sbox[counter2 & 0x000000FF]
        rc4Sbox[k] = rc4Sbox_value2
        rc4Sbox[(counter2 & 0x000000FF)] = rc4Sbox_value1
        tmp1 = rc4Sbox[((0x20 * counter1) ^ (counter2 >> 3)) & 0x000000FF]
        tmp2 = rc4Sbox[((0x20 * counter2) ^ (counter1 >> 3)) & 0x000000FF]
        tmp3 = rc4Sbox[((tmp1 + tmp2) & 0x000000FF) ^ 0xAA]
        tmp4 = rc4Sbox[(rc4Sbox_value2 + rc4Sbox_value1_) & 0x000000FF]
        tmp5 = (tmp3 + tmp4) & 0x000000FF
        tmp6 = rc4Sbox[(counter2 + rc4Sbox_value2) & 0x000000FF]
        decrypted.append(data[idx] ^ (tmp5 ^ tmp6))

        counter1 += 1
        j = counter1
        idx += 1

    return bytes(decrypted)
```

Below are the results of using a Python script to extract the configuration of Warzone RAT from the samples used in the article.



3. End

The article would like to conclude here. I hope that it provides useful information for you during the process of analyzing the Warzone RAT malware. To protect against Warzone RAT and other types of malware, users should take precautions such as being cautious when opening email attachments, using strong passwords, and keeping their software up to date. It is also important to use antivirus software and to keep it updated regularly. By taking these steps, users can help to protect themselves against the threat of Warzone RAT and other types of malware.

4. Refs

https://research.openanalysis.net/warzone/malware/config/2021/05/31/warzone_rat_config.html

https://exploitreversing.files.wordpress.com/2022/11/mas_6-1.pdf