New APT Group Earth Berberoka Targets Gambling Websites With Old and New Malware

By By: Daniel Lunghi, Jaromir Horejsi Apr 27, 2022 Read time: 10 min (2664 words)

Published: 2022-04-27 · Archived: 2026-04-05 17:55:24 UTC

We recently found a new advanced persistent threat (APT) group that we have dubbed Earth Berberoka (aka GamblingPuppet). This APT group targets gambling websites on Windows, macOS, and Linux platforms using old and new malware families.

We recently discovered a new advanced persistent threat (APT) group that we have dubbed Earth Berberoka (aka GamblingPuppet). Based on our analysis, this group targets gambling websites. Our investigation has also uncovered that Earth Berberoka targets the Windows, Linux, and macOS platforms, and uses malware families that have been historically attributed to Chinese-speaking individuals.

In this blog entry, we provide an overview of the Windows malware families used by Earth Berberoka in its campaign. This malware lineup includes tried-and-tested malware families that have been upgraded, such as PlugX and Gh0st RAT, and a brand-new multistage malware family that we have dubbed PuppetLoader.

We discuss the full technical details of Earth Berberoka's malware families aimed at Linux and macOS as well as Windows, infection vectors, targets, and possible connections with other APT groups in our research paper "Operation Earth Berberoka: An Analysis of a Multivector and Multiplatform APT Campaign Targeting Online Gambling Sites."

Technical analysis

PuppetLoader

We discovered a new malware family that we have dubbed PuppetLoader. It is a complex, five-stage malware family that uses some interesting techniques, including hijacking loaded modules to launch malicious code and hiding malicious payloads and modules in modified bitmap image (BMP) files.

Stage 1: Obfuscator and loader

Incorrect RC4 implementation

In this stage, a blob of payload data is decrypted using a hard-coded key (2726c6aea9970bb95211304705b5f595) and what appears to be an RC4 (Rivest Cipher 4) algorithm. However, the cipher's "swap" operation in the pseudorandom generation part of the cipher code was improperly implemented. This resulted in the proper

encryption of only the headers and the first few sections of the payload, while the latter sections were left almost entirely in clear text.

It seems that this hard-coded key and flawed RC4 implementation were also used in a malware family named TigerPlug, probably because it spreads the PlugX malware. We found no public reporting of its behavior and features.

Hijacking loaded module

After the payload is decrypted, it is loaded in the machine's memory and is executed using a stealthy method: PuppetLoader starts by loading a legitimate DLL from the Windows\System32 directory. The loading process is then hijacked to replace the legitimate library with a malicious one. This is done by hooking NTDLL APIs such as NtQueryAttributesFile, NtOpenFile, NtCreateSection, NtMapViewOfSection, NtQuerySection and ZwClose.

The loader uses undocumented NTDLL APIs, such as RtlPushFrame, RtlPopFrame, and RtlGetFrame, to avoid recursive booking, which happens when a hooked function indirectly calls itself.

To properly load the malicious payload, a frame tagged as "LDFM" is allocated and filled with the necessary parameters, such as file names' memory addresses and allocated buffer handles or addresses that contain the malicious payload. After their values are identified, some parameters are set immediately, while others are set at a later time.

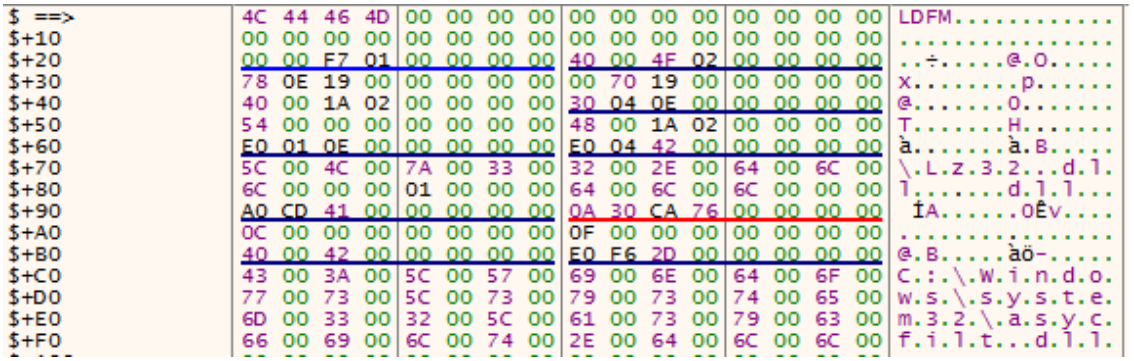

Figure 1. LDFM loading frame

LdrLoadDll is then called to load a legitimate asyclfilt.dll library. Afterward, previously hooked API functions are called, resulting in the loaded DLL name's being replaced with lz32.dll, which is a legitimate DLL. The content of this legitimate DLL is then replaced by a malicious payload that is inside the hooked NtMapViewOfSection function.

Then, the LdrLoadDll function rebases the newly loaded malicious image and loads all of the needed dependencies. The malware no longer needs the frame once the handle is returned from the LdrLoadDLL function, which is why it pops the frame by calling RtlPopFrame, unhooks previously hooked functions, and verifies whether the loading is successful or not by calling GetModuleHandleW (asycfilt.dll).

The malware then dynamically resolves the export function called Install and sets the parameter value to "11BF29E1371C0D83C530BD1BF346", which decrypts to a function called OneTime. For its command-line parameters, PuppetLoader uses the same flawed RC4 implementation using the password "whk0q9ogev6ofg8d".

These hijacking steps result in the following:

- The loaded asycfilt.dll module can be seen by parsing the PEB_LDR_DATA structure that contains all loaded modules in the current process.

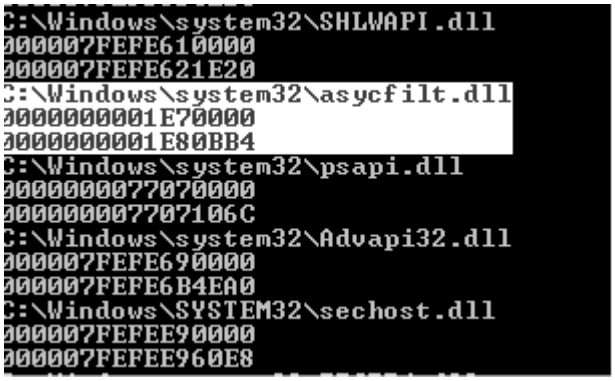

Figure 2. asycfilt.dll shown among loaded module names

- lz32.dll is opened based on process monitoring tools.

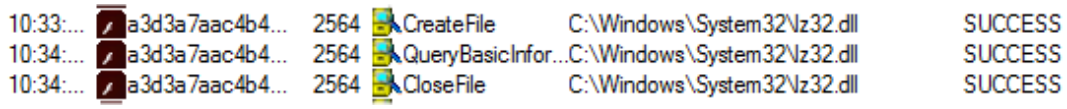

Figure 3. lz32.dll shown among opened files

- Only PuppetLoader's dropper payload is loaded and none of the previously mentioned libraries is actually loaded.

Stage 2: Dropper

The dropper creates and drops several files in an infected machine.

File	Function
CpuppetProcessFileSharer	Used for sharing data during the different infection stages
Config.ini	Saves the execution reason and the globally unique identifier (GUID) value based on ComputerName
MSVCPX00.dll	DLL file of BasicLoader
verisign.bmp	BMP file with encrypted Core
bitmap.bmp	BMP file with encrypted Client.MainConsole

Table 1. The files created and dropped by the dropper

The hard-coded GUID ({78106D5F-CD1A-A8C4-A625-6863092B4BBA}) is inserted into CPuppetProcessFileSharer (C:\\Users\\Public\\Pictures\\Desktop.inf). We believe that it serves as a marker that

stage 2 has been completed.

Config.ini (C:\Users\Public\Videos\Config.ini) contains the GUID and the reason, which is the hard-coded value "StartupBasicLoader" encrypted using a key (whk0q9ogev6ofg8d).

The svchost.exe is started in suspended mode with this command-line parameter:

-cmd -NoModuleLoadDLL -DisplayName=KeepAuthority.Client.MainConsole.x64.Release -InvokeMethodName=Run -InokeMethodParam=NULL"

This is also encrypted with the previously mentioned key and a new thread is created within svchost.exe to make it load the BasicLoader payload, MSVCPX00.dll. It is interesting to note that there is a typographical error in "-InokeMethodParam."

Stage 3: BasicLoader

The BasicLoader stage starts by adding a hard-coded GUID ({78106D5F-CD1A-A8C4-A625-6863092B4BBA}) into CPuppetProcessFileSharer. As with stage 2, we believe that this is likely a marker that stage 3 has started running.

BasicLoader searches for BMP files across directories in Users\\Public (Desktop, Documents, Downlaods, Music, Pictures, and Videos). It checks each directory for BMP files that would pass the required structure. For the BMP files that do, the payload appended to the BMP file is decrypted, loaded into memory, and executed. The BMP file is made up of only 33 x 11 pixels and 338 bytes, and the data appended to it is the payload that is encrypted with the same flawed RC4 implementation.

Figure 4. A small BMP file, to which encrypted payloads are appended

Stage 4: Core

This stage begins when a hard-coded GUID ({7D8DA9DC-1F3B-2E5C-AA59-9418E652E4AA}) is added to CPuppetProcessFileSharer. Similar to the other stages, this is likely a marker that stage 4 has started running.

After this, the malware starts a system logger thread, where the logged information is received via a pipe and is saved to a file with a hard-coded name. The logged information can come from other modules or processes. Based on our analysis, each log file entry is separated by a separator (0xAABBCCDD), followed by a custom RC4 password and message length.

The decrypted log can include the following information:

- The module that was run
- The parameters at which this module was run
- At which stage (GUID from CPuppetProcessFileSharer) the action was performed

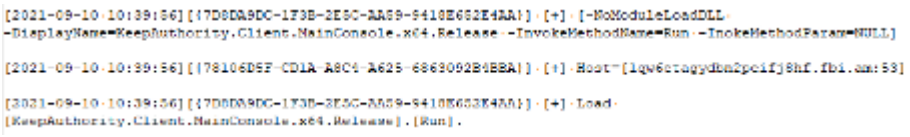

Figure 5. Decrypted log from Core

The following command-line arguments are also implemented during this stage:

- -DisplayName
- -InokeMethodParam (sic)
- -InvokeMethodName
- -NoModuleLoadDLL
- -LoadShellcode

While the other arguments are mostly self-explanatory, we highlight two arguments worth noting: -NoModuleLoadDLL uses the same technique as the stage 1 loader and -LoadShellcode allocates a memory block, copies shellcode, and executes it.

Stage 5: Client.MainConsole

This is the main client binary written in C++, which is the last stage of PuppetLoader's infection chain.

The code is structured in several classes that handle different tasks, such as managing the interactive shell, uploading and downloading files, installing new modules, monitoring victim behavior, and executing callback functions when conditions are met.

- CPipeCmdManager – interactive shell manager

Arguments:

-flushusersession

-createcmd

-destorycmd (sic)

-excutecmd

-cmdkeepalive

- CommonLib::CcmdMulArgDecoder – command-line argument decoder, additional module related to command-line arguments

Arguments:

-ModuleLog

-LogText

-ModuleID

-ModuleVersion

-MountStatus

-Path

-IsDelete

-ModuleKeepAlive

-UploadFile

The client then establishes communication with a command-and-control (C&C) server via UDP (User Datagram Protocol) and recognizes different types of custom UDP packets:

UDP packet	Description
RemoteModuleCommandPacket	Command to be executed by interactive shell
RemoteModuleCommandResultPacket	Result of running shell command
FileTransferContent_Packet	Determines whether to upload or download a file
UploadFilePacket	Uploaded file content
FileManage_FolderContent_Packet	Folder content
VecProcessPacket	Vector object with running processes
InstallModulePacket	BMP with encrypted module to be installed
RemoteClientSystemInfoPacket	Sent by login callback every time a new user logs in
ModuleKeepAlivePacket	Tells the C&C server that the connection is still alive

Table 2. The custom UDP packets recognized by the client

The backdoor functions implemented in the main client are:

- Interactive shell
- Upload file
- Download file
- List files
- Terminate process
- List processes
- Install module
- Login callback
- Enumerate RDP sessions

The communication protocol via UDP uses the same RC4 encryption. A sent and/or received packet contains a 16-byte RC4 key and the length of an RC4-encrypted payload, followed by another packet with the encrypted payload itself.

oRAT

Another malware family that we obtained both Windows and macOS samples of during our investigation was oRAT. Interestingly, this was the first time that we had analyzed samples of this malware family written in the Go language.

The oRAT droppers that we found in our analysis were a MiMi chat application built using the Electron JS framework and a DMG (disk image) file. We discuss the full details of both in our research paper.

The samples are flagged as version 0.5.1 for both Windows and macOS samples, and have the same features and configurations.

The configuration file and the AES decryption key are appended in an encrypted form to the PE (Portable Executable) file overlay.

```
{
    "Local": {
        "Network": "sudp",
        "Address": ":5555"
    },
    "C2": {
        "Network": "stcp",
        "Address": "darwin.github.wiki:53"
    },
    "Gateway": false
}
```

Figure 6. The decrypted oRAT configuration

The configuration is decrypted using the AES-GCM (AES with Galois/Counter Mode) algorithm. The malware then parses it and enables the gateway or traffic forwarder mode if it is specified in the configuration settings.

For the malware operator to directly connect to an infected machine and execute commands via GET or POST requests, the malware starts local servers on the infected machine to listen on ports that have been specified in the configuration settings for control commands.

The network communications can be in plain text or encrypted, depending on the configuration of the file:

- “tcp” for plain text
- “stcp” for encrypted TCP communications using the golang-tls library
- “sudp” for encrypted UDP traffic using the Quic-go library

The control server is implemented by registering routes. This simple mechanism leads to translating GET/POST requests directly as internal Go commands. Requesting a URL therefore results in executing the corresponding code on an infected system.

We obtained oRAT samples that register these routes:

- GET /agent/info
- GET /agent/ping
- POST /agent/upload
- GET /agent/download
- GET /agent/screenshot
- GET /agent/zip
- GET /agent/unzip
- GET /agent/kill-self
- GET /agent/portscan
- GET /agent/proxy
- GET /agent/ssh
- GET /agent/net

PuppetDownloaders (C++ downloaders)

During our investigation, we also saw malicious websites that distribute fake Adobe Flash Player updates that were actually delivering C++ downloaders.

The infection starts with an executable written in C++ that connects through a Winsock API to a domain or IP address in a specific port. The downloaded content is saved as SMTemp.dat, and using the executable's file name and a hard-coded XOR key, a file named Loader.dll is decrypted and copied to the disk. If the executable is renamed for whatever reason, the DLL decryption fails and the malware's second stage does not go through.

If the SMTemp.dat file exists, the Loader.dll file executes it. After that, the loader decrypts a legitimate Adobe Flash Player installer and executes it, in order to deceive the victim into thinking that the executable is a legitimate installer.

During our investigation, we noted that the server hosting the second stage of the PuppetDownloaders malware payload was offline. It is also interesting to note that the string decryption routine of this malware is a simple XOR with the string "2020-05-24 13:00:29" as its key. The first 13 bytes of the password that is used to decode the string are the same as the last 13 bytes.

```
_WORD *__fastcall decode_string(_WORD *encoded, _WORD *decoded)
{
  *decoded = *encoded ^ '2';
  decoded[1] = encoded[1] ^ '0';
  decoded[2] = encoded[2] ^ '2';
  decoded[3] = encoded[3] ^ '0';
  decoded[4] = encoded[4] ^ '-';
  decoded[5] = encoded[5] ^ '0';
  decoded[6] = encoded[6] ^ '5';
  decoded[7] = encoded[7] ^ '-';
  decoded[8] = encoded[8] ^ '2';
  decoded[9] = encoded[9] ^ '4';
  decoded[10] = encoded[10] ^ ' ';
  decoded[11] = encoded[11] ^ '1';
  decoded[12] = encoded[12] ^ '3';
  decoded[13] = encoded[13] ^ ':';
  decoded[14] = encoded[14] ^ '0';
  decoded[15] = encoded[15] ^ '0';
  decoded[16] = encoded[16] ^ ':';
  decoded[17] = encoded[17] ^ '2';
  decoded[18] = encoded[18] ^ '9';
  decoded[19] = encoded[19];
  decoded[20] = encoded[20] ^ '2';
  decoded[21] = encoded[21] ^ '0';
  decoded[22] = encoded[22] ^ '2';
  decoded[23] = encoded[23] ^ '0';
  decoded[24] = encoded[24] ^ '-';
  decoded[25] = encoded[25] ^ '0';
  decoded[26] = encoded[26] ^ '5';
  decoded[27] = encoded[27] ^ '-';
  decoded[28] = encoded[28] ^ '2';
  decoded[29] = encoded[29] ^ '4';
  decoded[30] = encoded[30] ^ ' ';
  decoded[31] = encoded[31] ^ '1';
  decoded[32] = encoded[32] ^ '3';
  return decoded;
}
```

Figure 7. XOR decryption routine

We dubbed these downloaders PuppetDownloaders since they are connected to the PuppetLoader malware family, as evidenced by our observations:

- This malware and PuppetLoader both use the same string decryption routine that uses the same key.
- This malware and PuppetLoader both use the same XOR key (2726c6aea9970bb95211304705b5f595) that is used to decrypt the embedded Loader.dll file.
- This malware and PuppetLoader's decrypted Loader.dlls share similar strings such as "[-] UnExist pwszModuleFunName:". This suggests that a common framework was used to compile both DLLs.

MFC socket downloaders

We also saw WinRAR self-extracting (SFX) files dropping downloaders written using the Microsoft Foundation Class Library (MFC) framework. These MFC socket downloaders have an identical structure: One function creates a socket, connects to a domain or IP address, sends a short string, and then calls "recv" twice.

The code flow is redirected through a call to EnumDesktopsA or EnumWindows, whose callback function pointers point to the downloaded content.

The downloaders attempt to access ports 8080, 29527, and 8885. They also send the strings "feiji", "@5436", and "fhfgj@jfggdsg" to the sockets. We found multiple additional samples of the same malware family that have the same structure and send the same strings. However, it is possible that multiple groups might be covertly sharing the source code for this malware.

PlugX

PlugX is a remote access tool (RAT) that has been used as a malicious tool for espionage for more than a decade. We found that Earth Berberoka uses PlugX to target 32-bit and 64-bit architectures, based on the samples we obtained and analyzed.

This malware family sends a DWORD, a 32-bit unsigned integer, in the HELLO packet. A compromised system then sends the HELLO packet, which looks like a date in the "yyyymmdd" format, to the C&C server.

We found the following DWORDs in multiple samples we analyzed, which suggest that the versions we found were developed within the last three years: 20190520, 20201106, and 20210804.

All of the samples we found are loaded in the same way: A legitimate and signed file that is vulnerable to DLL sideloading is placed alongside a malicious DLL, which decrypts and loads the third file containing the final payload.

One of these malicious DLL files has the PDB (program database) path C:\Users\Administrator\Desktop\Plug7.0(Logger)\logexts\x64\Release\logexts.pdb.

Gh0st RAT

We also saw at least three different variants of Gh0st RAT, another malware family that has been in the wild for more than 10 years, being used in Earth Berberoka's campaign. This malware family's source code is public, which is why it has many variants.

One of the variants we analyzed had an interesting destructive feature: It replaces the master boot record (MBR) to display an explicit message ("I am virus ! F*ck you :-)"). This particular message was also seen in a public report from a victim of this Gh0st RAT variant. A 2017 Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) report also discussed how Gh0st RAT variants wiped the MBR and replaced it with messages that varied across different samples.

Other Known Malware Families

We also found other legitimate tools being abused by Earth Berberoka and a malware family being used by the group in its campaign:

- Quasar RAT – a Windows-based open-source RAT that has been used by APT groups for network exploitation
- AsyncRAT – an open-source RAT that can be used to remotely monitor and control devices via an encrypted connection
- Trochilus – a stealthy RAT that can evade sandbox analysis and can be used in cyberespionage campaigns

Security recommendations

Our analysis points to Earth Berberoka's having multiple tools and a large infrastructure at its disposal to target the Southeast Asian gambling market. To avoid falling victim to Earth Berberoka's attacks, users and operators of gambling websites can adopt the following security recommendations:

- Properly vet emails, websites, and apps before clicking on links or downloading apps.
- Download apps only from trusted sources.
- Watch out for malicious website flags, such as errors in grammar and spelling.

- Block threats that arrive via email, such as malicious links, through hosted email security and antispam protection.
- Use a multilayered security solution that helps with detecting, scanning, and blocking malicious URLs.

The full technical details of our investigation can be found in our research paper, which we will publish soon. We list down the indicators of compromise (IOCs) for Windows, Linux, and macOS in separate text files. We also provide the domain list in a separate text file.

Tags
