DarkVision RAT | ThreatLabz

By Muhammed Irfan V A

Published: 2024-10-10 · Archived: 2026-04-05 22:23:17 UTC

Technical Analysis

The following sections offer a technical analysis of an attack chain used to deploy DarkVision RAT, as well as an in-depth examination of the RAT itself.

The figure below illustrates the attack chain for the DarkVision RAT campaign discussed in this blog.

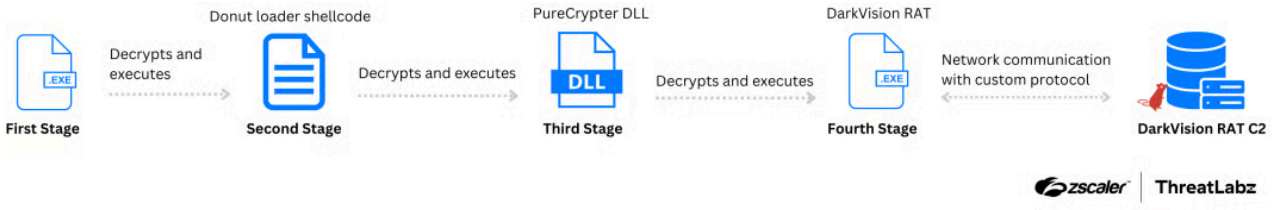

Figure 1: An example attack chain distributing DarkVision RAT as the payload in the final stage.

First stage: Shellcode decryption in DarkVision RAT attack

The initial stage in this attack chain is a .NET executable file, protected using .NET Reactor. Upon execution of the .NET file, the first stage runs the following command:

```
cmd /c timeout 10
```

After the brief 10-second delay, the .NET file moves on to its next phase, where it decrypts the second stage shellcode.

The .NET executable uses Triple Data Encryption Standard (3DES) to decrypt the second stage shellcode. The **key** and **IV** are encoded in Base64 format. The Base64-encoded strings are `xwmyVxHV39B5ns41HJtzRQ==` for the **key** and `SzD5abWvrRk=` for the **IV**. The .NET executable file decodes these strings back into their original binary form. The decoded key and IV are then fed into the 3DES algorithm to decrypt the shellcode.

The decrypted shellcode is written to a block of memory that is made executable using `VirtualAlloc` and `VirtualProtect`. The .NET executable then uses the `API EnumCalendarInfo`'s callback function to execute the shellcode leading to the second stage.

Second stage: Donut loader

The decrypted second stage shellcode is the open source Donut loader. This x86 position-independent shellcode is designed to load .NET assemblies directly into memory. Donut loader uses the Chaskey block cipher to encrypt its

modules.

We won't be covering the specifics of Donut's loading process, as several excellent write-ups already exist on the topic. Instead, to proceed with our analysis, we used Donut Decryptor to extract the third stage payload.

Third stage

Loading DarkVision RAT with PureCrypter

The third stage of the attack chain is a .NET assembly, identified as PureCrypter, which has been previously analyzed by ThreatLabz. The main function of the PureCrypter injector starts by decompressing (`gunzip`) and deserializing an object into a `protobuf` structure, as shown in the figure below.

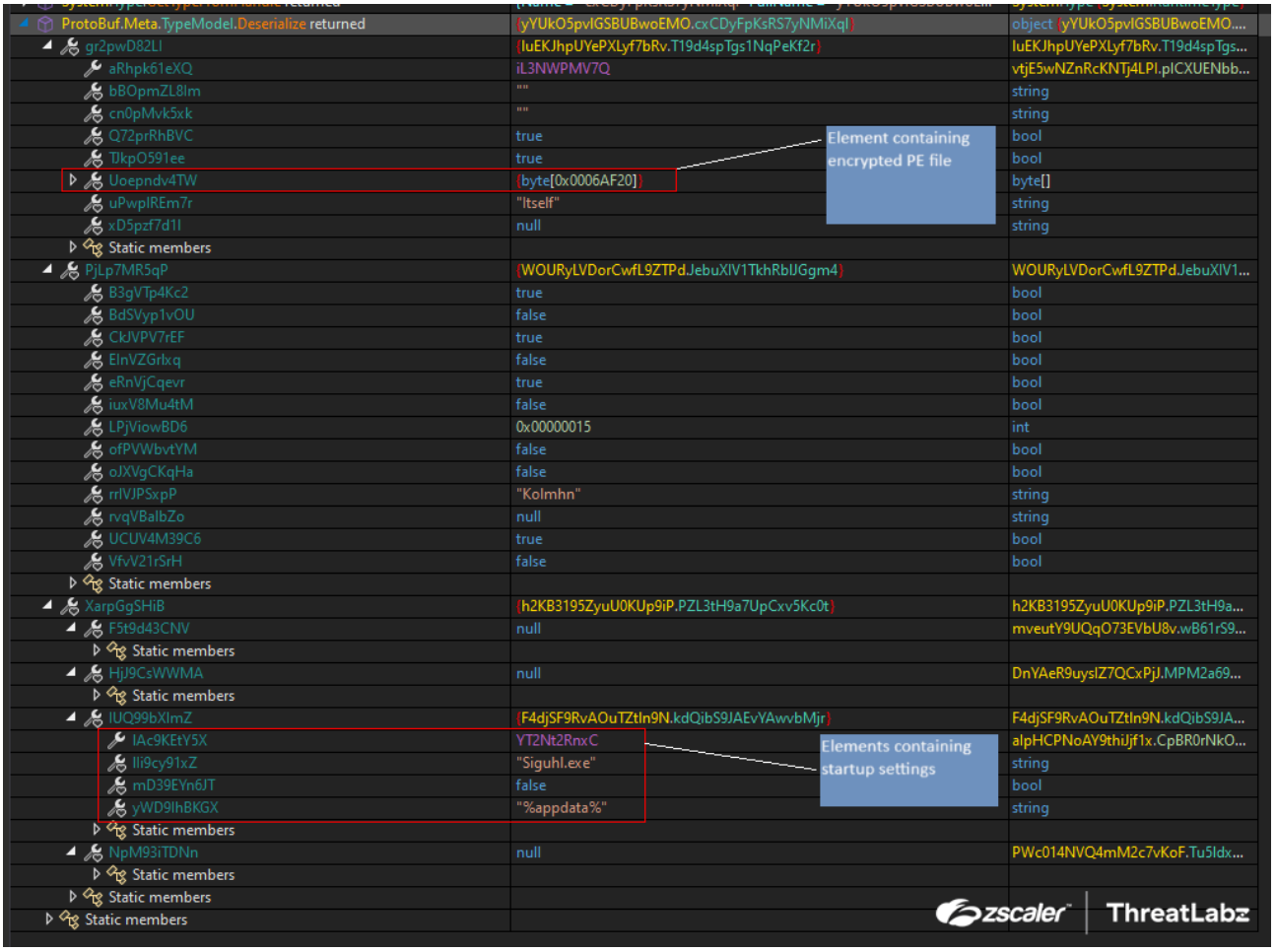

Figure 2: PureCrypter `protobuf` structure.

One of the key members in this `protobuf` structure is `gr2pwD82LI` which contains an element named `Uoepndv4TW` . This particular element holds the DarkVision RAT payload portable executable (PE) content, which is encrypted using Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode.

Another important part of the `protobuf` structure is a member named `IUQ99bXImZ` , which contains the startup settings for DarkVision RAT.

Windows Defender exclusion and persistence tactics in PureCrypter

PureCrypter executes a PowerShell command that has been encoded in Base64 format. When decoded, this command tells PowerShell to add malicious file paths and process names used by the RAT to the list of exclusions in Windows Defender. The example below shows the PowerShell commands used to add Windows Defender exclusions for malicious file paths and process names used by DarkVision RAT.

```
 Add-MpPreference -ExclusionPath C:\yknoahdrv.exe;
Add-MpPreference -ExclusionProcess yknoahdrv.exe;
Add-MpPreference -ExclusionPath C:\Users\REDACTED\AppData\Roaming\Siguhl.exe; Add-MpPreference -ExclusionProcess
```

PureCrypter doesn't stop at evading detection as it also helps DarkVision RAT achieve persistence. PureCrypter writes the current file to `%APPDATA%\Sighul.exe` and adds persistence for this file as per the `protobuf` struct by using the Auto-run registry key `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` with the name set to `Sighul`. Then, according to the values in the `protobuf` struct, the decrypted DarkVision RAT file is injected into itself (current process), and execution is transferred to the entry point of DarkVision RAT, leading to the fourth stage.

Fourth stage: Persistence and C&C protocol

DarkVision RAT first dynamically resolves APIs using `GetProcAddress` and `LoadLibrary`. The RAT always reloads the libraries again using `LoadLibrary` to avoid userlands hooks placed by antivirus and endpoint detection and response (EDR) software. The API names used by the malware are stored in XOR-encoded form and are decoded using the XOR key `[19 72 19 72]`. DarkVision RAT also uses XOR encoding to store important strings. From here, DarkVision RAT starts to parse the command-line arguments.

Command-line parsing in DarkVision RAT

After decoding the necessary strings, DarkVision RAT starts to parse any command-line arguments. The command-line arguments used by the DarkVision RAT are Globally Unique Identifiers (GUIDs). These GUIDs serve as names in various places such as registry keys, folder names, and file names. When investigating other samples of DarkVision RAT, we noticed that these GUIDs differ from one sample to another, indicating a level of randomness in each instance of the RAT ensuring these cannot be used to create detection logic for DarkVision RAT.

Here are two GUIDs and how they were used in this sample:

- **{B8B1DC5F-E2FC-41FF-A2D1-DB3800909230}**:

Conditions: The action below is carried out if the user is not a local administrator and the Windows version is greater or equal to 10.

Action: Under these conditions, DarkVision RAT attempts to gain elevated privileges using a technique called DLL hijacking. DarkVision targets `WinSAT.exe`, a legitimate Windows process, and `DXGI.DLL`, a dynamic link library file to attempt auto elevation.

- **{14C43BB8-A5DF-4F5D-A77A-E8BB32DEE41F}**:

Conditions: The Actions below are carried out if the user is a local administrator and the Windows version is greater or equal to 10.

Action: In this scenario, DarkVision RAT adds an exclusion rule to Windows Defender to avoid detection. The RAT achieves this by running the command `cmd.exe /c powershell.exe Add-MpPreference -ExclusionPath`, which tells Windows Defender to ignore the RAT file path.

Adding DarkVision RAT data to the Windows registry

DarkVision RAT creates a registry key under HKEY_CURRENT_USER\SOFTWARE\ and adds three values, each named using a hardcoded GUID. The values stored are the following:

1. **RAT file content**: The command opcodes 0x2BD and 0x2BE (discussed later) use this value to write the RAT file to disk.
2. **RAT file path**: Based on a flag, the RAT deletes the file stored in this file path. This flag is related to deleting artifacts.
3. **Current system time stored in a FILETIME structure**: This value is stored in a FINGERPRINT_INFO1 struct (discussed later) which is sent to the C2 server.

The figure below shows the data being added to the Windows registry.

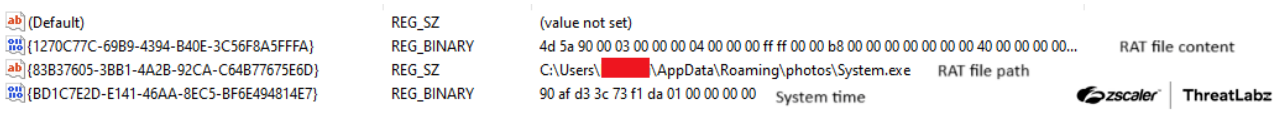

Figure 3: DarkVision RAT data added to the Windows registry.

Persistence mechanisms leveraged by DarkVision RAT

DarkVision RAT employs three different methods to ensure persistence on an infected system. Like most of the features in DarkVision RAT, there are flags for each persistence technique, which store a boolean value that decides which persistence mechanism should be used in the sample. Since these flags are hardcoded into the binary, we concluded that they are configurable options available to the attacker when a DarkVision RAT sample is created using a builder.

The three persistence methods are as follows:

- **Startup folder** - In this method, DarkVision RAT creates a batch script that contains a command to execute the RAT executable. After creating this script, DarkVision RAT then creates a shortcut to the batch script and places this shortcut in the Windows startup folder.

- **Autorun keys** - Another method DarkVision RAT leverages is autorun keys. DarkVision RAT adds an entry that points to its executable file in one of the autorun keys located at Software\Microsoft\Windows\CurrentVersion\Run. The exact location of this key can be under

either HKEY_CURRENT_USER (for the current user) or HKEY_LOCAL_MACHINE (for the system), depending on the flags set within the RAT.

- **Task Scheduler -** DarkVision RAT uses the ITaskService COM interface to schedule a task to execute the malware.

After setting up persistence mechanisms, DarkVision RAT checks if it is currently running from a specific location, namely `%APPDATA%\photos\System.exe` (we refer to `%APPDATA%\photos` as the RAT folder and the full path as the RAT file path moving forward as this path varies across samples).

If DarkVision RAT is not running from this designated folder, it copies itself to this path and then uses the newly copied file to create a child process. This ensures that the RAT is running from a consistent and expected location.

Next, the RAT creates a new folder in `C:\ProgramData`, which we refer to as the plugin parent folder. This folder is used to store additional encrypted plugins in its subdirectories (discussed later).

Process injection techniques employed by DarkVision RAT

DarkVision RAT uses the `NtCreateSection` and `NtMapViewOfSection` APIs to perform process injection, which is used in multiple places to perform RAT functionalities. DarkVision RAT creates a remote process in a suspended state. The RAT then creates a new memory section, mapping one view of this section to the local process and another view to the remote target process. The view mapped in the local process is populated with a function that the RAT needs to execute. This process is repeated to fill the structure used by the function in another mapped view. The thread context of the remote process is then modified: the Instruction Pointer (RIP/EIP) is set to the function's address, and the first parameter (RCX/ESP+4) is set to the address of the structure. Finally, the thread is resumed, leading to the execution of the function.

DarkVision RAT communication protocol

Once executed, DarkVision RAT needs to connect to the C2 server to receive instructions and respond with information. The C2 communications use a custom binary protocol. Based on the flags set, the C2 address is parsed in one of two different ways:

- **Retrieve the C2 information**: The RAT utilizes WinHTTP libraries to connect to a URL stored in plain text. The returned data contains the C2 information in the format `c2address:port`.
- **Hardcoded C2 Information**: The C2 address and port are stored in plain text within the binary. For example, the C2 address embedded in the binary analyzed by ThreatLabz was `severdops.ddns[.]net:8120`.

Registration

The first action DarkVision RAT takes is to register itself with the C2 server by sending a unique Bot ID. To create this Bot ID, the RAT generates a random GUID and combines it with an MD5 hash of the Unicode string `"P@55w0rd!"`. This string, `"P@55w0rd!",` is stored in plain text within the RAT's code and varies across different samples.

Receiving the acknowledgment (ACK) packet

After sending its unique ID, DarkVision RAT waits for a response from the C2 server. The server replies with a specific data packet `{ 01 00 00 00 }`. The received data is compared to the value `1`, and the RAT will only proceed with sending the next data if this comparison is successful. This packet functions similarly to an ACK packet in the TCP protocol, so we will refer to it as an "ACK packet" moving forward. The RAT client then sends the data `{ 00 00 00 00 }`, to which the server responds with an ACK packet.

The figure below shows the network communication between a system infected with DarkVision RAT and the C2 server.

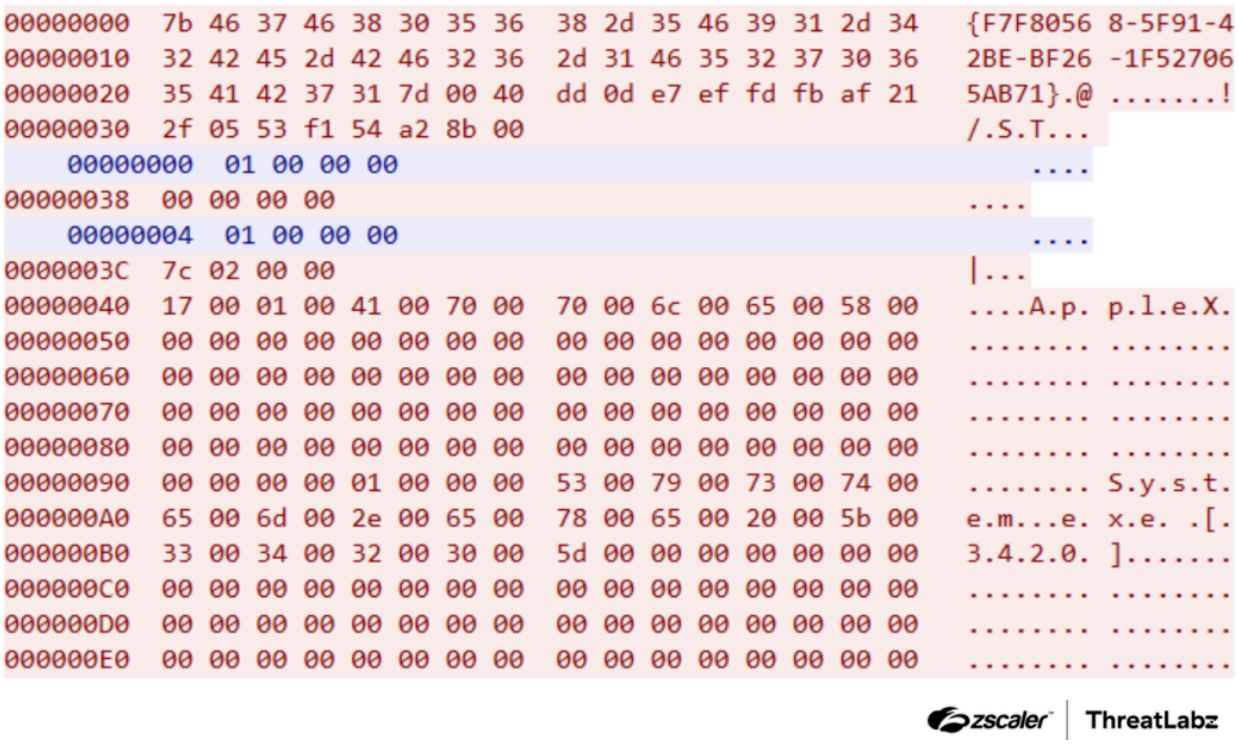

Figure 4: Network communication between a system infected with DarkVision RAT and the C2 server.

Device fingerprinting

The RAT client then performs device fingerprinting and collects system information. This information is sent in two packets. Before each packet, the size of the structure is sent, followed by the structure containing the system information. After receiving the first structure, the server sends an ACK packet. Upon receiving the second structure, the C2 server sends two ACK packets. The two structures sent are shown below.

```
struct FINGERPRINT_INFO1{
uint32_t hardcoded_value;           // set to 0x10017
wchar_t botnet_name[40];            // set to AppleX
int32_t is_localadmin;              // TRUE = 1 , FALSE = 0
wchar_t pname_followedby_pid[260];  // RAT process name followed by pid %s [%d]
uint32_t hardcoded_value2;          // set to 0x40
```

```
 FILETIME system_time;              // current system time
};
```

```
 struct FINGERPRINT_INFO2{
 uint32_t geocode;            // geocode of the victim country
 wchar_t computer_name[16];   // victim computer name
 wchar_t user_name[258];      // victim username
 uint32_t system_ip[54];      // victim h_addr_list in network byte order
 int64_t system_uptime;       // victim system uptime in milliseconds
 ULONG os_info[4];            // victim os info
};
```

DarkVision RAT then creates a new socket and sends its unique bot ID to the server. The server responds with an ACK packet. The client then sends an ACK packet in return, and the server replies with another ACK packet. After this exchange, the RAT client waits for commands from the C2 server.

Commands supported by DarkVision RAT

The command's opcode, function address, and other related data are stored as an array of 12 elements (12 commands). Each element is a struct of size 0x28, which we will refer to as a COMMAND_STRUCT. If the opcode matches the data received from the server, the corresponding func_address is executed by creating a thread. The COMMAND_STRUCT is shown below.

```
 struct COMMAND_STRUCT{
 uint64_t opcode;        // opcode of the command
 uint64_t event_handle;  // handle of event created when the command is run
 uint64_t thread_handle; // handle of thread created which executes func_address
 void *func_address;     // address of the function to be executed
 uint64_t socket;        // socket descriptor of the socket
};
```

The table below lists the commands supported by DarkVision RAT.

Opcode	Description
0x2BD	Writes the RAT file content stored in the Windows registry to the RAT's file path.
0x2BE	Writes the RAT file content stored in the Windows registry to the RAT's file path and executes it.

Opcode	Description
0x2BF	Receives a file from the C2 server via socket, writes it to disk, executes it, and then deletes all RAT artifacts, including persistence entries from the registry and file system.
0x2C0	Deletes all RAT artifacts, including persistence entries from the registry and files from the disk.
0x2C1	Runs the RAT executable as an administrator using the `runas` verb. If this attempt fails, create a RAT process that does not require administrator permissions.
0x2C2	Performs DLL hijacking via `WinSAT.exe` and `DXGI.DLL` to achieve auto-elevation.
0x2C3	Receives a URL and user agent from the C2 server via socket. Downloads the file from the URL using the user agent provided, writes the file to disk, executes it, and then deletes all RAT artifacts.
0x519	Receives a compressed plugin from the C2 server, decompresses it using LZNT1, and loads it into memory. Encrypts the plugin with Salsa20 (the key and nonce are hardcoded) and writes it to disk. The C2 server sends data in the following order: plugin ID, plugin compressed size, and plugin data in compressed form.
0x51A	Unloads the specified plugin ID. The plugin ID follows the data received from the C2 server after the opcode.
0x51B	Retrieves the status of all plugins (whether loaded or not).
0x51C	Deletes the encrypted plugin from the disk and registry based on the data received from the C2 server. If the data received is the value `1`, deletes all plugins. Otherwise, the RAT deletes the specified plugin ID received from the C2 server.
0x51D	Receives the specified plugin ID from the C2 server. Encrypts it using Salsa20 (using the same hardcoded key and nonce used in opcode 0x519), then writes it to both the registry and the disk.

Table 1: Commands implemented by DarkVision RAT.

There is another set of commands used to execute plugin ordinals. For all previously mentioned commands, the upper 16 bits are set to `0`, and the lower 16 bits contain the opcode ID of the command to be executed. In the next set of commands, the upper 16 bits contain the plugin ID to be executed, while the lower 16 bits contain the ordinal number to be executed.

Plugins available in DarkVision RAT

Most of the DarkVision RAT's features are implemented through plugins. These plugins remain as plain text only in memory, while they are stored as encrypted data on disk and in the registry. When a plugin is loaded, ordinal 0x65 of the plugin is executed using a thread. The thread takes a struct as an argument, which contains important information about the plugin. Below is the structure used for this purpose.

```
 struct PLUGIN_STRUCT{
 wchar_t plugin_parent_folder[0x8000]; // folder containing all plugin sub folders
 wchar_t plugin_filename[0x8000];      // plugin file name
 wchar_t plugin_filepath[0x8000];      // plugin file path
 void *plugin_base_address;            // plugin base address in memory
 int32_t plugin_size;                  // size of plugin
 void *rat_folder;                     // RAT folder
 void *plugin_array;                   // array containing all plugin id's
};
```

The plugins are executed in the same manner as the initial set of commands. The table below shows the plugin ID and its description.

Plugin ID	**Plugin Description**
0x1	Captures webcam footage.
0x2	Displays messages using MessageBox.
0x3	Retrieves the process list and terminates processes based on PID.
0x4	Edits the registry.

Plugin ID	Plugin Description
0x5	Provides file system access.
0x6	Views victim screen via screenshots.
0x7	Lists and manages system windows.
0x8	Performs system control activities such as locking the workstation, shutting down, or restarting.
0x9	Retrieves and sets desktop wallpaper.
0xD	Establishes a reverse proxy using SOCKS.
0xE	Acts as a dropper to download a file from a URL and write it to disk.
0xF	Open a remote shell.
0x10	Captures microphone audio.
0x11	Records keystrokes live (live keylogger).
0x12	Steals passwords.
0x13	Provides remote access using VNC.
0x14	Provides remote access using hVNC.

Plugin ID	Plugin Description
0x15	Records keystrokes offline (offline keylogger).
0x16	Locks the workstation or shut downs the system for protection when the victim is away from the keyboard.
0x17	Retrieves the process list and creates a minidump of processes based on PID.

Table 2: Plugins loaded by DarkVision RAT.
