Mustang Panda: PAKLOG, CorKLOG, and SplatCloak | ThreatLabz

By Sudeep Singh, ThreatLabz

Published: 2025-04-16 · Archived: 2026-04-05 13:23:37 UTC

Technical Analysis

The technical analysis in this section focuses on the keyloggers, PAKLOG and CorKLOG, along with SplatCloak. Both PAKLOG and CorKLOG are straightforward keyloggers, but CorKLOG includes persistence mechanisms. Additionally, both keyloggers obfuscate log files to conceal their activity. SplatCloak, deployed by SplatDropper, is a Windows kernel driver designed to disable EDR-related routines implemented by Windows Defender and Kaspersky, enabling it to evade detection.

PAKLOG

Paklog is a keylogger that utilizes high-level Windows APIs to capture keystrokes and monitor clipboard activity. The keylogger then encodes this data and writes it to a local file. Paklog lacks built-in exfiltration capabilities, suggesting that Mustang Panda mainly uses it to collect data and leverages other methods for data exfiltration. Paklog is deployed via a RAR archive (e.g., `key.rar`), which contains two files: a signed, legitimate binary (`PACLOUD.exe`) and the malicious Paklog DLL (`pa_lang2.dll`). The `PACLOUD.exe` binary is used to sideload the Paklog DLL which starts the keylogger functionality. The sections below provide more information about Paklog.

The Paklog DLL (`pa_lang2.dll`) includes a malicious export function named *ASH_LANG2_add*, which performs the following actions:

- **Keylogging Setup:** Utilizes the *SetWindowsHookExW* API, with `idHook` set to `WH_KEYBOARD_LL`, and a custom hook procedure.
- **Data Logging:** The custom hook procedure tracks keystrokes within the foreground window and logs the information in the following format:
  - **timestamp:** The precise time the key was pressed, formatted as `%Y-%m-%d %H:%M:%S`.
  - **window_text:** The text of the foreground window.
  - **process_full_path:** The full path of the process active in the foreground.
- **Key Mapping:** Each key's virtual code is mapped to a specific string that represents the corresponding key pressed by the user.

In addition to keylogging, Paklog can be used to monitor clipboard activity and extract its contents. Whenever the user presses the `Ctrl + V` key combination, the malware intercepts the request, retrieves the clipboard data, and

copies it to a buffer. Both the keystrokes and clipboard data are encoded and saved to the file: `C:\\Users\\Public\\Libraries\\record.txt`.

PAKLOG uses a simple encoding mechanism to encode the characters in the buffer.

Uppercase characters

If the ASCII code of the character is below 0x5a (the uppercase character range), the new character code is determined using the formula:

```
new_char_code = (orig_char_code - 0x3e % 0x1a) + 0x41
```

Lowercase characters

If the ASCII code of the character is above 0x5a (the lowercase character range), the new character code is determined using the formula:

```
new_char_code = (orig_char_code - 0x5e % 0x1a) + 0x61
```

CorKLOG

CorkLOG is another keylogger designed to capture keystrokes, storing the captured data in an encrypted file using a 48-character RC4 key. To ensure it runs continuously, the keylogger establishes persistence on the system by creating services or scheduled tasks. CorkLOG is delivered through a RAR archive (e.g., `src.rar`), which contains two files: an executable (`lcommute.exe`) and the CorKLOG DLL (`mscorsvc.dll`). Mustang Panda likely intended on the executable being used to sideload the DLL to activate the keylogger. However, due to an error, the executable does not sideload the DLL, since the executable does not load a DLL with CorKLOG DLL's filename.

The CorKLOG DLL achieves persistence through one of two mechanisms, depending on its privilege level. If it detects that it is running with administrator privileges, the malware installs itself as a service. The second persistence mechanism creates a scheduled task called *TabletInputServices*. The scheduled task is set to run every 10 minutes using the following command-line:

```
schtasks /create /tn TabletInputServices /tr /sc minute /mo 10 /f
```

CorKLOG then executes schtasks a second time to run the task immediately.

The keylogging code records its output to a file, encrypting the contents with RC4 using the key `fkpioefpoea$@^Tf0-0-gepwf09IJGEJ0IFAPKO456SG894E` before writing them to disk.

CorKLOG includes a configuration file stored in the `.config` section of the binary. While the configuration file for this sample is missing, our analysis suggests that it typically contains the following values:

- Service name
- Service display name
- Folder

The code snippet below illustrates the multi-stage decryption process applied to the configuration values. This process uses the same XOR key, but begins at four different offsets within the key, performing a series of XOR operations on the encrypted string.

```
 void crypto_xor_decrypt(PCSTR buffer, int32_t size)
{
   uint8_t index = 0;
   if (size)
   {
       // First decryption pass, the xor key starting location is at the beginning of the xor key
       do
       {
           // xor key length := 0x3f
           buffer[index] ^= xorkey[index & 0x3f];
           index += 1;
       } while (index
```

SplatDropper

SplatCloak (discussed later) is deployed by SplatDropper, a small utility that drops the driver onto the system, executes it, and then removes itself to avoid leaving traces. SplatDropper is delivered inside a RAR archive (e.g., `kb.rar`), which contains two files: a legitimate executable (`BugSplatHD64.exe`) and the SplatDropper DLL (`BugSplat64.dll`).

The legitimate executable sideloads the SplatDropper DLL, which performs the following actions:

1. Resolves Windows APIs by hash.
2. Decrypts the kernel driver (SplatCloak) using a single-byte XOR key (0x5a) and writes the result to disk.
3. Creates a Windows service to execute SplatCloak.
4. Waits 5 seconds to allow SplatCloak to run.
5. Stops the Windows Service.
6. Cleans up (removes the Windows service and deletes SplatCloak).

SplatDropper resolves the required Windows API calls by hash. The hash algorithm is identical to the methods used in tonepipeshell and tonepipeshell_alt, but with a seed value of 131313 (instead of previously observed values that include 13131313 and 1313131313). The API hashing algorithm is shown in the code snippet below:

```
 char* apiname_1 = apiname;
int32_t api_hash = 0;
while ((int32_t)*(uint8_t*)apiname_1)
{
   api_hash = api_hash * 131313 + (int32_t)*(uint8_t*)apiname_1;
   apiname_1 = &apiname_1[1];
```

```
}
return (uint64_t)api_hash;
```

SplatDropper generates three random strings, consisting of characters from A-Z, to create the filename, service name, and service display name. The algorithm is depicted in the code snippet below.

```
 // Generate a char buffer containing the ascii codes for A-Z
char alphabet[0x1a];
for (int32_t i = 0; i
```

SplatCloak

The sole purpose of the SplatCloak driver is to identify and disable, or remove notification hooks and callbacks associated with Windows Defender and Kaspersky. SplatCloak makes use of a revoked certificate (certificate thumbprint: 09ededdcbdb0c03c850f1d29920e412348120c8d) issued to Xtreaming Technology Inc. with the start date of 2010-02-22 00:00:00 UTC and the end date of 2012-02-22 23:59:59 UTC. The threat actor is exploiting the fact that Windows allows drivers with revoked certificates to be loaded. ThreatLabz also confirmed that this driver will be successfully loaded on a fully patched Windows 10 system. Other interesting features implemented in SplatCloak are control flow flattening and mixed boolean arithmetic used to hinder reverse engineering efforts by malware analysts.

The SplatCloak driver, similar to its dropper (SplatDropper), dynamically resolves Windows API functions; however, the code does not utilize API hashing. Instead, the driver resolves API calls by retrieving the SYSTEM_MODULE_INFORMATION structure via *ZwQuerySystemInformation* (with the `SystemInformationClass` parameter set to `SystemModuleInformation`) and traversing the returned SYSTEM_MODULE_ENTRY structure to locate the base address of ntoskrnl.exe. This process is shown in the figure below.

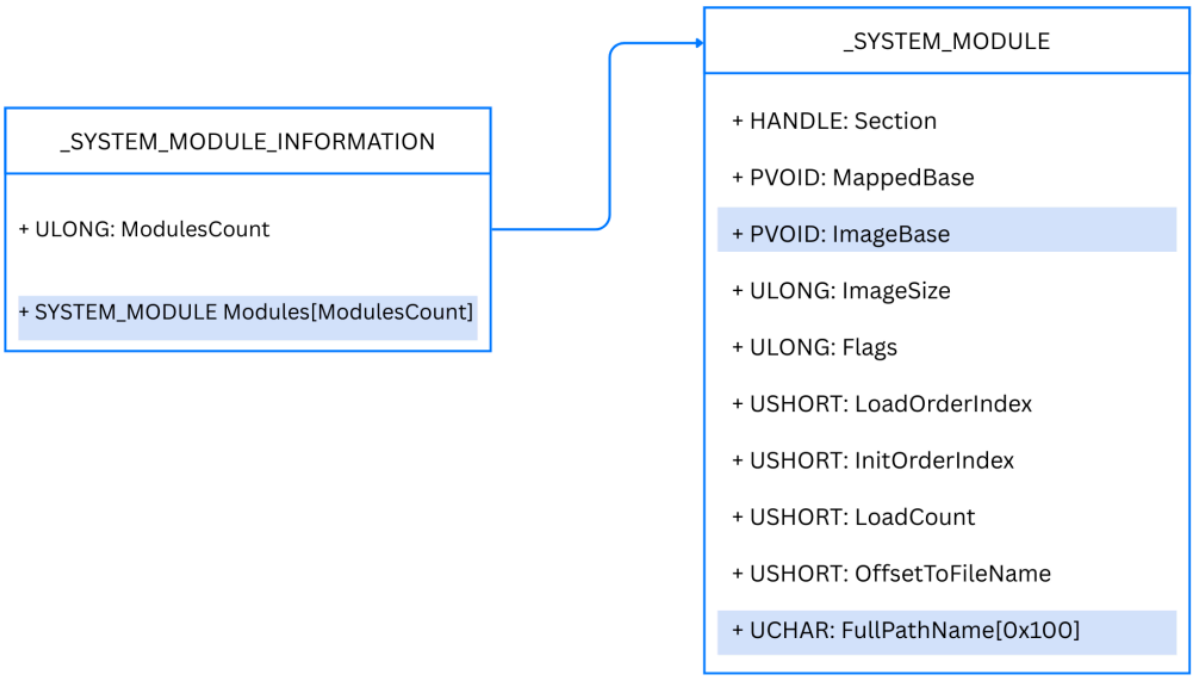

Figure 1: Relationship between the system module information structure and the system module structure.

Using the base address of ntoskrnl, the SplatCloak driver parses its Portable Executable (PE) structure to obtain the address of *MmGetSystemRoutineAddress*. This function is used to resolve various kernel-level Windows APIs, including the locations of *PsProcessType* and *PsThreadType*, which are also exported by ntoskrnl. Additionally, *RtlGetVersion* is resolved to retrieve the Windows build number.

The SplatCloak driver retrieves the callback routine list pointers supplied by the system, which are shown in the table below:

List Name	**List Length**	**Description**
PspCreateProcessNotifyRoutine	64	List of function pointers that are executed when a process is created.

List Name	List Length	Description
PspCreateThreadNotifyRoutine	64	List of function pointers that are executed when a thread is created.
PspLoadImageNotifyRoutine	64	List of function pointers that are executed when an image is loaded into memory.

Table 1: List of kernel-level notification routines identified by the SplatCloak driver.

This retrieval is accomplished by analyzing the bytes of the resolved functions to identify a specific pattern that points to their associated notification list of function pointers. The logic for each routine is illustrated in the figure below.

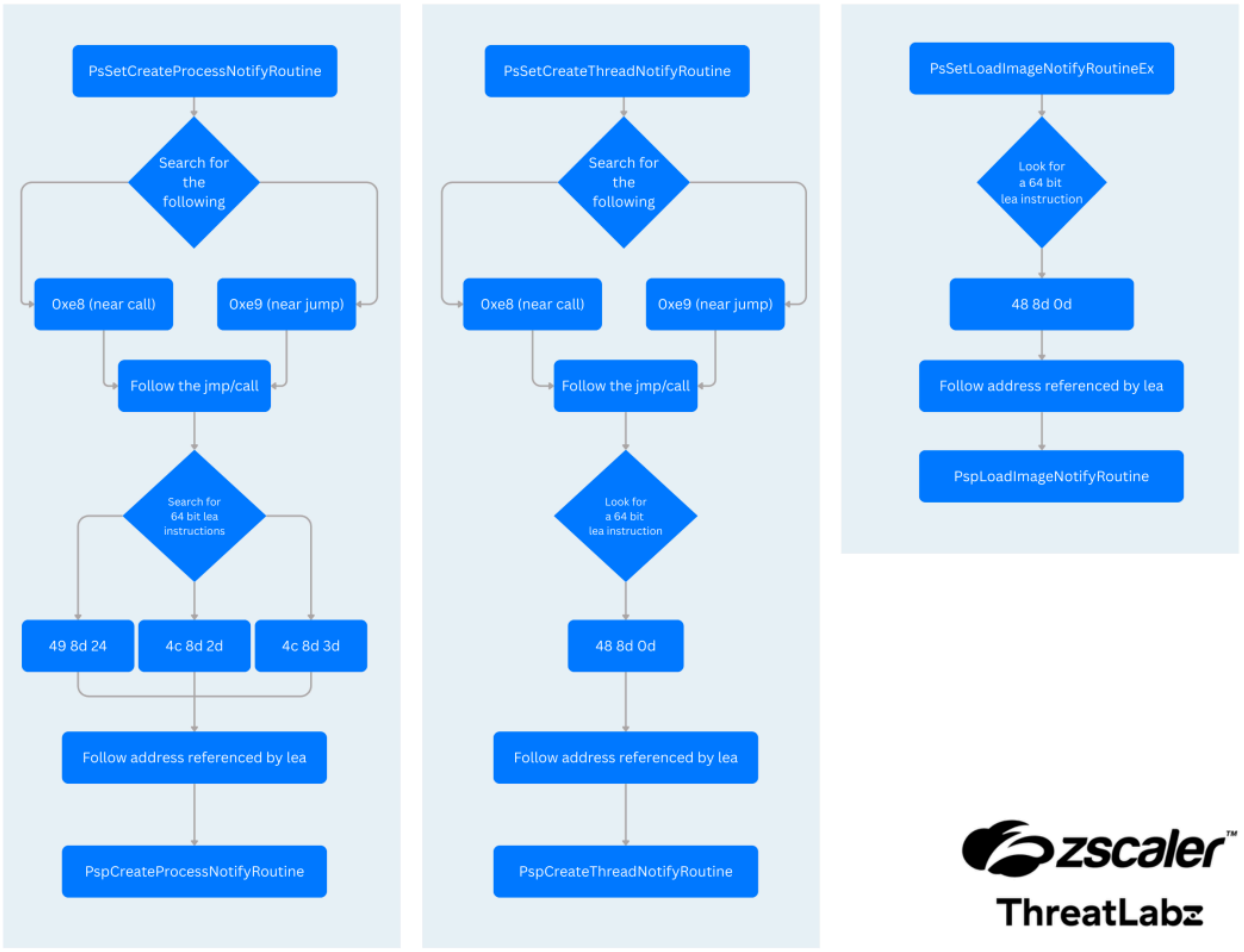

Figure 2: SplatCloak logic for locating the routines: (left to right) *PspCreateProcessNotifyRoutine*, *PspCreateProcessNotifyRoutine*, and *PspLoadImageNotifyRoutine*.

The CallbackListHead (registry related callbacks) location is also determined by inspecting the bytes in *CmUnRegisterCallback*, as shown in the figure below.

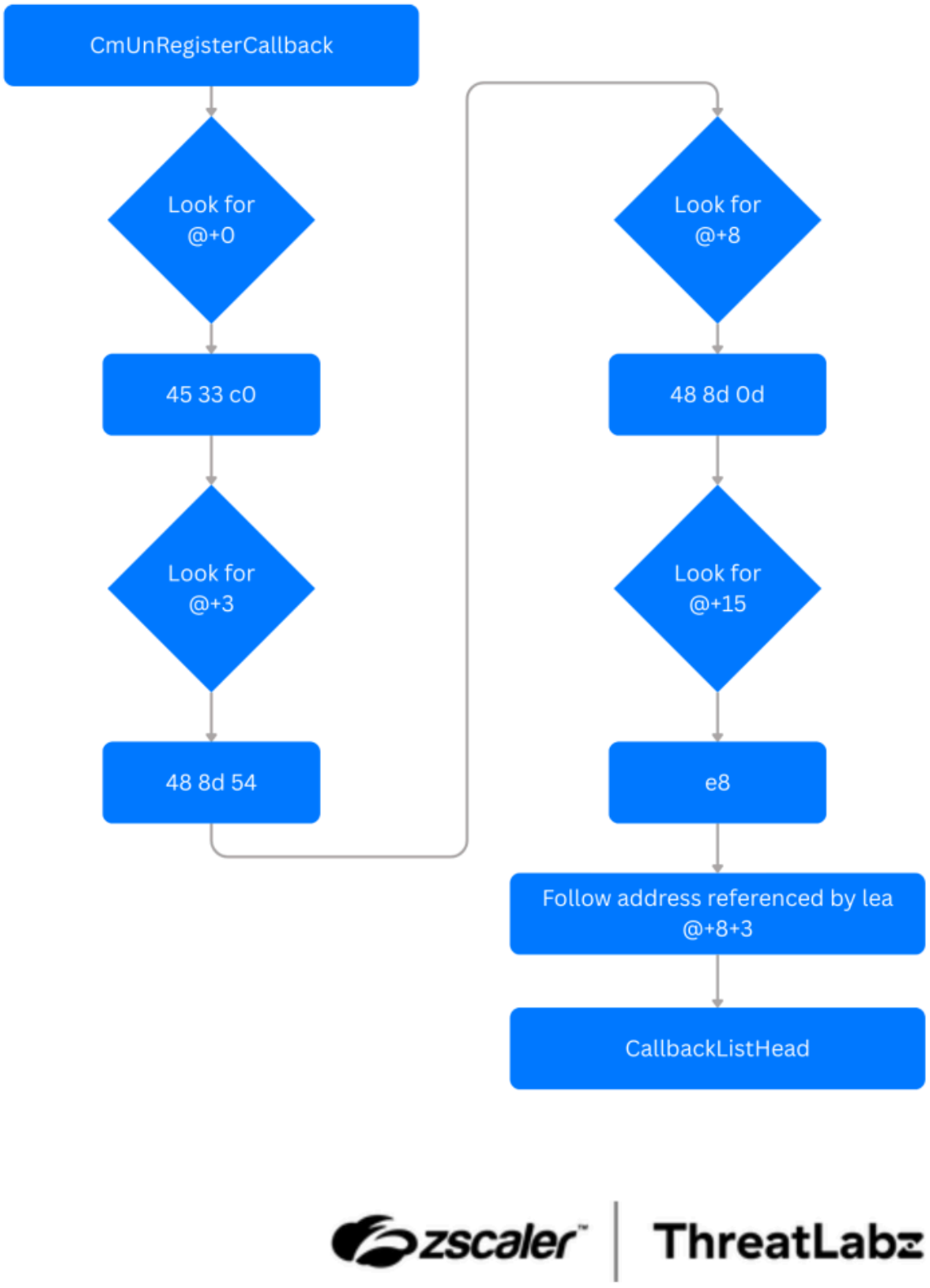

Figure 3: Shows the process used by the SplatCloak driver to find the CallbackListHead location.

If the Windows `dwBuildNumber` retrieved via *RtlGetVersion* is 19000 or higher (Windows 10 version 2004 or later), the driver proceeds to unregister and disable the identified callbacks. However, it's worth noting that even in newer Windows versions with build numbers above 19000, the driver may not function correctly, as the opcodes for inspected functions have changed.

To identify whether a callback or notification routine should be disabled, the SplatCloak driver retrieves the FullPathName from `_SYSTEM_MODULE_ENTRY` using *ZwQuerySystemInformation* as previously described. The malware then examines the filename to determine if it matches any of the Windows Defender-related drivers listed in the table below.

Driver	**Description**
wdfilter.sys	Responsible for monitoring and filtering file system activity to detect and prevent malicious behavior.
wdboot.sys	Operates during system boot to ensure protection against threats before the operating system fully loads.
wddevflt.sys	Designed to monitor and filter device-level operations, providing an additional layer of security for hardware interactions.
wdnisdrv.sys	Inspects and filters network streams to detect and block potential threats transmitted over the network.

Table 2: List of Windows Defender-related files monitored by SplatCloak.

The Kaspersky check involves enumerating each callback and notification routine within the system to pinpoint the binary associated with the function they reference. Once the corresponding binary is identified, the SplatCloak driver maps the binary into memory and analyzes its structure to locate the `IMAGE_DIRECTORY_ENTRY_SECURITY`. This entry provides a pointer to the code signing certificate embedded within the binary.

Unlike other entries in the `IMAGE_DATA_DIRECTORY` array, which typically use relative virtual offsets, the `IMAGE_DIRECTORY_ENTRY_SECURITY` uses a physical file offset to indicate the location of the signed certificate. This physical offset points to where the certificate is stored, which is typically found in the file overlay — the data appended to the binary beyond the standard PE file structure. This process is shown in the figure below.

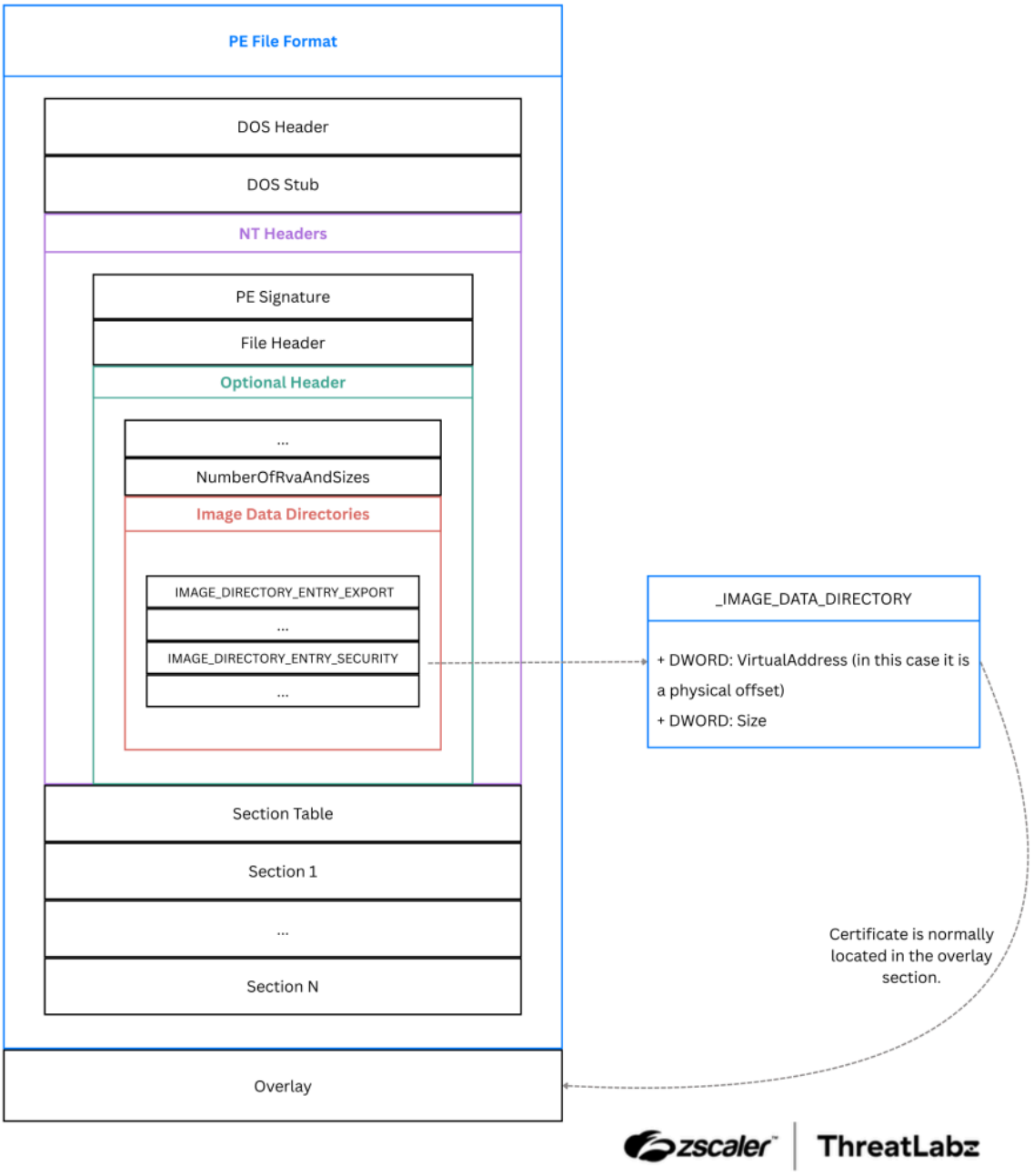

Figure 4: SplatCloak process for determining the location and size of the code signing certificate.

The SplatCloak driver uses this offset, if present, to locate the certificate and examine the associated bytes for the keyword *kaspersky*.

With the exception of callbacks related to *PsProcessType* and *PsThreadType*, any callbacks or notifications that match either Kaspersky or drivers associated with Windows Defender are removed using the appropriate APIs.

- *PsSetCreateProcessNotifyRoutine*
  - The `remove` parameter is set to `True`.
- *PsSetCreateProcessNotifyRoutineEx*
  - The `remove` parameter is set to `True`.
- *PsSetCreateProcessNotifyRoutineEx2*

    - The `remove` parameter is set to `True`.
- *PsRemoveCreateThreadNotifyRoutine*
- *PsRemoveLoadImageNotifyRoutine*
- *CmUnRegisterCallback*

For *PsProcessType* and *PsThreadType*, the `OB_CALLBACK_ENTRY_t` structure's `Enabled` value is set to `FALSE`, effectively disabling the callback. Both *PsProcessType* and *PsThreadType* are of type `_OBJECT_TYPE`. These structures and their relationships are depicted in the figure below.

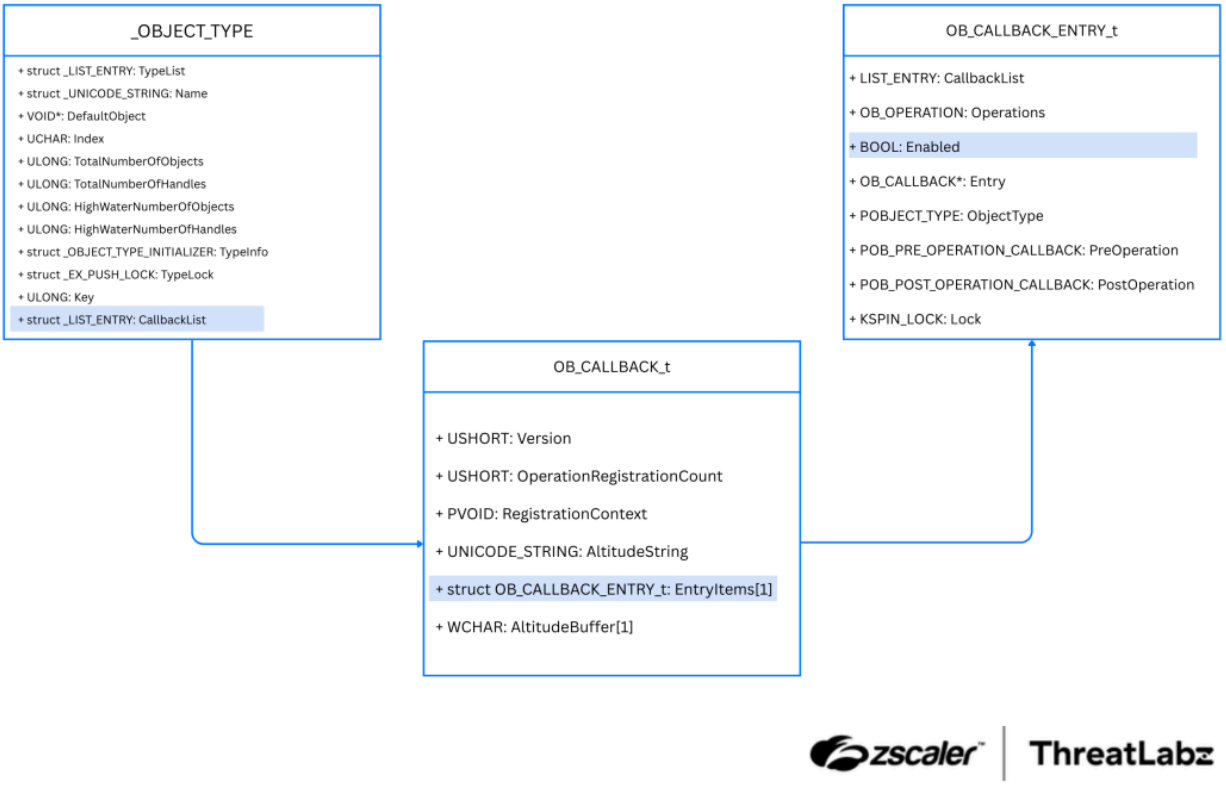

Figure 5: Shows path from _OBJECT_TYPE structure to the OB_CALLBACK_ENTRY_t structure. (Reference: https://github.com/wavestone-cdt/EDRSandblast)

Explore more Zscaler blogs
