Tech Analysis of Rhadamanthys Obfuscation Techniques | Blog

By Nikolaos Pantazopoulos, Sarthak Misraa

Published: 2023-02-21 · Archived: 2026-04-05 12:59:18 UTC

Technical Analysis

The following subsections focus on the technical analysis of the Rhadamanthys components.

Loader

The loader consists of different stages until the actual loader starts its execution. We have categorized these stages as follows:

- Initialization Phase
- Decompression Phase
- Loader Phase

Initialization Phase

During the initialization phase, Rhadamanthys main task is to decode an embedded block and pass the execution there. In addition, it detects and passes to the next phase the following information:

1. Encrypted configuration
2. A compressed blob that contains modules for assisting with code injection and the in-memory loader

In general, we have identified two different types of loaders. Interestingly in one of them, Rhadamanthys uses a virtual machine (Q3VM) in order to obfuscate its code and hide certain code details.

Each virtualized block of the protected code is executed by passing an integer value as a parameter to the interpreter of the virtual machine. The identified features of the protected code are summarized in Table 1 below.

Parameter	Code Description
0	Decodes the next phase using the Base32 algorithm with the custom charset A-Z4-9=
1	Loads the Windows API functions *GetProcAddress* and *VirtualProtect* by using the ROR-13 hashing technique.
2	Calls the loaded *VirtualProtect* Windows API function to prepare the shellcode for execution.
3	Gets a set of strings and searches for them in the current’s process memory space. These strings are:

	i) avast.exe ii) snxhk

Table 1 - Rhadamanthys Virtualized functions

Additionally, we identified a sample, which includes a de-virtualized version of the last code block (parameter 3) and the PDB path:

```
d:\debugInfo\rhadamanthys\debug\sandbox.pdb
```

NOTE: The magic bytes of the VM bytecodes have been modified by the threat actors as an attempt to hide the usage of the tool that was used. Moreover, in more recent samples, they have added the XTEA algorithm as an additional layer of encryption for the decoded payload.

Decompression Phase

In the second phase, the decoded shellcode loads dynamically a set of Windows API functions and decompresses the loader's code using the LZSS algorithm.

Loader Phase

In the final stage, the loader decrypts its configuration using the RC4 algorithm and proceeds with the download process of the main module. The structure of the decrypted configuration is the following:

```
 struct config
{
  unsigned int Magic;
  unsigned int Flag; // Used during command line parsing since version 0.4.1
  unsigned char Key_Salt[0x10]; // Used during the AES decryption of the downloaded main module.
  unsigned char C2[]; // The URL path to download the main module. The main module uses the same path for data e
};
```

It is worth to note that the final stage of the loader has its own header structure, which is described below. The information derived from this structure is necessary for the loader in order to apply necessary code relocations.

```
 struct Loader_Header
{
  unsigned __int16 Magic; // Set to 52 53
  unsigned __int16 Characteristics;
  unsigned __int16 Sections_Number;
  unsigned __int16 Sizeof_Header;
  unsigned int Entry_Point_Offset;
  unsigned int Stager_Size;
  unsigned int Imports_Offset;
```

```
  unsigned int  Imports_Size;
  unsigned int  Unknown1;
  unsigned int Unknown2;
  unsigned int Relocation_Table_Offset;
  unsigned int Relocation_Table_Size;
  section Stager_Sections[5];
};
```

```
 struct section
{
  unsigned int Disk_Section_Offset;
  unsigned int Rva_Section_Offset;
  unsigned int Section_Size;
};
```

Embedded File System

When Rhadamnthys compromises a 64-bit host, the loader decompresses (LZMA) an embedded file system. The embedded file system includes several modules that aim to assist the execution process of the main module. The structure of the file system and its embedded modules along with a description of them (Table 2) are mentioned below.

```
 struct loader_embedded_vfs
{
  unsigned char hardcoded_value; // Set to 0xB
  unsigned char num_of_modules;
  unsigned __int16  base_Address;
  module_info modules[num_of_modules];
};

struct module_info
{
  unsigned int module_hash; // MurmurHash. Used to detect the module.
  unsigned char module_size_offset; // The byte is left shifted with the value 0xc.
};
```

Module Name	Description
prepare.bin	Applies relocations and dynamic API loading.
dfdll.dll	Executable file written on disk. It loads and executes the downloaded payload.
unhook.bin	Detects if specified Windows API functions of NTDLL library have been hooked.
phexec.bin	Injects code by using the SYSENTER command while calling Windows API functions.

Table 2 - Identified embedded modules

NOTE: In case of a 32-bit compromised host, none of the above modules are required. Instead, Rhadamanthys generates a key by doing a bitwise XOR operation of the first byte of the downloaded module with the hard-coded byte value 0x21. The output is used as an XOR key to decrypt the first 108 bytes (header) of the downloaded payload

Main module

Similarly with the loader component, the main module has its own set of modules and components. As can be seen in Table 3, the main module has a variety of embedded components.

Module Name	Description
KeePassHax	C# module to exfiltrate credentials of password management software KeePass.
Stubmod	Assists with communication between modules and Coredll by using a named PIPE.
Stub	Loads and executes the main module from disk.
Coredll	Main orchestrator.
Preload	Executes Coredll.
Runtime	C# module to execute PowerShell scripts.
Stubexec	Module, which injects code to another process (regsvr32).
/etc/license.key	Unknown. Potentially related to a license key.
/etc/puk.key	Elliptic Curve (NIST P-256) public key
/extension/%08x.lua	A set of LUA scripts, which are used for extracting credentials.

Table 3 - Main module embedded components

Furthermore, instead of using hardcoded offsets to detect and extract them, Rhadamanthys uses the MPQ hashing algorithm to hash the name of the embedded component and generate a set of hashes. Then, it uses these hashes to scan its own memory in order to detect the appropriate component.

Explore more Zscaler blogs
