r77 Rootkit

Technical Documentation



|  |  |
| --- | --- |
| r77 Version | 1.7.1 |
| Release date | 27.08.2025 |
|  |  |
| Author | bytecode77 |
| Website | [bytecode77.com/r77-rootkit](https://bytecode77.com/r77-rootkit) |
| GitHub | [github.com/bytecode77/r77-rootkit](https://github.com/bytecode77/r77-rootkit) |

Table of Contents

[1 Abstract 4](#_Toc207218213)

[2 User’s Manual 5](#_Toc207218214)

[2.1 Install.exe 5](#_Toc207218215)

[2.2 Uninstall.exe 5](#_Toc207218216)

[2.3 Test Console 5](#_Toc207218217)

[2.4 Example.exe 7](#_Toc207218218)

[2.5 Hidden Entities 7](#_Toc207218219)

[2.5.1 File System 8](#_Toc207218220)

[2.5.2 Processes 9](#_Toc207218221)

[2.5.3 Registry 9](#_Toc207218222)

[2.5.4 Services 10](#_Toc207218223)

[2.5.5 TCP & UDP Connections 10](#_Toc207218224)

[2.5.6 Windows 11](#_Toc207218225)

[2.6 Configuration System 11](#_Toc207218226)

[2.6.1 Process ID’s 12](#_Toc207218227)

[2.6.2 Process Names 12](#_Toc207218228)

[2.6.3 Paths 12](#_Toc207218229)

[2.6.4 Service Names 12](#_Toc207218230)

[2.6.5 Local TCP Ports 12](#_Toc207218231)

[2.6.6 Remote TCP Ports 12](#_Toc207218232)

[2.6.7 UDP Ports 12](#_Toc207218233)

[2.6.8 Startup Paths 12](#_Toc207218234)

[3 Integrator’s Manual 13](#_Toc207218235)

[3.1 Install.exe 13](#_Toc207218236)

[3.2 Install.shellcode 13](#_Toc207218237)

[3.3 Control Pipe 13](#_Toc207218238)

[3.4 Enumeration vs. Direct Access 15](#_Toc207218239)

[3.5 Custom Startup Files 15](#_Toc207218240)

[4 Implementation Details 17](#_Toc207218241)

[4.1 Compilation 17](#_Toc207218242)

[4.2 Rootkit DLL 17](#_Toc207218243)

[4.3 r77 Service 17](#_Toc207218244)

[4.3.1 Fileless Startup 17](#_Toc207218245)

[4.4 Child Process Hooking 19](#_Toc207218246)

[4.5 Shellcode Installation 19](#_Toc207218247)

[4.6 Dependencies & Requirements 20](#_Toc207218248)

[4.6.1 Elevated Privileges 20](#_Toc207218249)

[4.7 Hooked API’s 20](#_Toc207218250)

[4.7.1 NtQuerySystemInformation 20](#_Toc207218251)

[4.7.2 NtResumeThread 20](#_Toc207218252)

[4.7.3 NtQueryDirectoryFile 21](#_Toc207218253)

[4.7.4 NtQueryDirectoryFileEx 21](#_Toc207218254)

[4.7.5 NtQueryKey 21](#_Toc207218255)

[4.7.6 NtEnumerateKey 21](#_Toc207218256)

[4.7.7 NtEnumerateValueKey 21](#_Toc207218257)

[4.7.8 NtUserBuildHwndList 21](#_Toc207218258)

[4.7.9 EnumServiceGroupW 21](#_Toc207218259)

[4.7.10 EnumServicesStatusExW 21](#_Toc207218260)

[4.7.11 NtDeviceIoControlFile 21](#_Toc207218261)

[4.7.12 PdhGetRawCounterArrayW 22](#_Toc207218262)

[4.7.13 PdhGetFormattedCounterArrayW 22](#_Toc207218263)

[4.7.14 AmsiScanBuffer 22](#_Toc207218264)

[4.8 AV Evasion Techniques 22](#_Toc207218265)

[4.8.1 AMSI Bypass 22](#_Toc207218266)

[4.8.2 DLL Unhooking 23](#_Toc207218267)

[4.9 r77 Header 23](#_Toc207218268)

[4.10 Compile Time Constants 24](#_Toc207218269)

[4.11 De-implementation of features 25](#_Toc207218270)

[5 Notes 26](#_Toc207218271)

[5.1 Supported Platforms 26](#_Toc207218272)

[5.2 Compatibility 26](#_Toc207218273)

[5.3 Tested Applications 26](#_Toc207218274)

[5.4 Known Issues 27](#_Toc207218275)

[5.5 ToDo List 27](#_Toc207218276)

[5.6 Bug Reports 28](#_Toc207218277)

[6 Change Log 29](#_Toc207218278)

# Abstract

r77 Rootkit is a fileless ring 3 rootkit. Its primary purpose is to hide files, directories, processes, services, registry entries, etc.

Moreover, the rootkit ships with an installer that persists r77 on the operating system. The installation is completely fileless, meaning no files are written to the disk. r77 solely relies on in-memory operations and remains in the system memory at all times. The installer’s persistence mechanism allows it to continue running after Windows reboots.

For the deployment of r77, only a single executable is required that needs to be executed only once.

# User’s Manual

This chapter explains:

* How to install and uninstall the rootkit
* How to test the rootkit on individual processes without affecting the entire system
* The general functionality of the rootkit
* How to configure the rootkit

## Install.exe

Run Install.exe to inject r77 into every running process and to persist the rootkit on the system. From this point forward, new processes are injected before they run any of their own instructions. This is achieved by hooking process creation. After installation, r77 is set up to start after reboot and inject all processes before the first user is logged on.

Install.exe already bundles all required files, therefore it is not necessary to deploy the DLL’s along with it. This is a **single file deployment**.

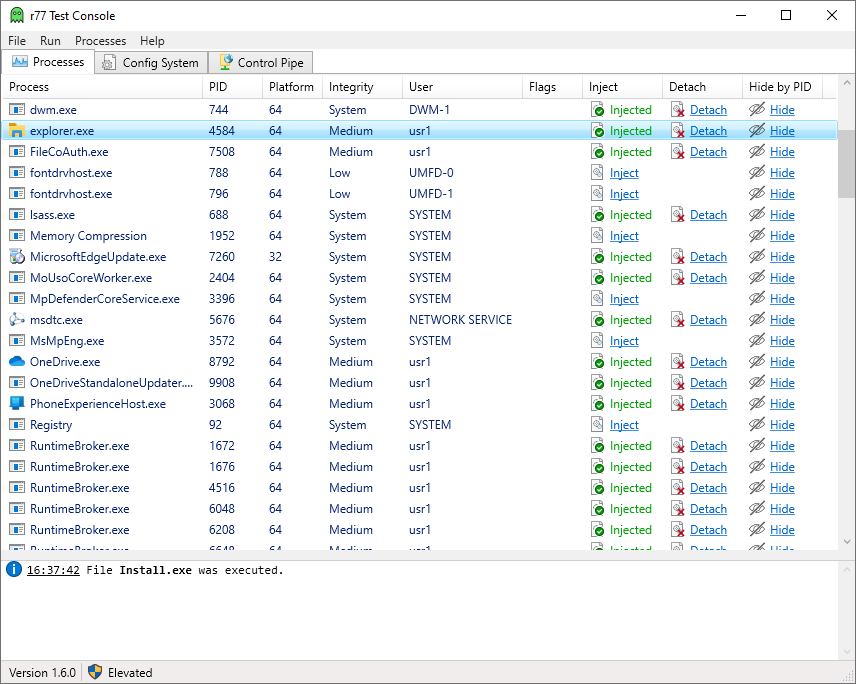
When executing Install.exe after r77 was already installed, the r77 service is terminated and restarted. This is supported behavior and the correct way of upgrading r77 to the current version. Already injected processes will **not** be detached and re-injected with the current version of the rootkit DLL. To do that, execute Uninstall.exe prior to executing Install.exe.

## Uninstall.exe

To remove r77 from the system completely, run Uninstall.exe. It will uninstall r77, detach the rootkit from all running processes and delete the r77 configuration from the registry. A reboot is not required.

## Test Console

TestConsole.exe is a tool for testing the functionality of r77. It can be used to inject r77 to or detach r77 from individual processes without installing the rootkit. However, some features are only available when the rootkit is fully installed.



The process list shows, which processes are injected. The “Flags”-Column shows additional information about a process.

|  |  |
| --- | --- |
| Flag | Meaning |
|  | This is the process that hosts the r77 service (winlogon.exe). It cannot be injected with r77.  The r77 service is running when r77 is installed. |
|  | This is an r77 helper process. It cannot be injected with r77.  TestConsole.exe is a helper process.  Also, Install.exe and Uninstall.exe are helper processes. This is to avoid r77 being injected into them. |
|  | This process is hidden by ID using the configuration system. A task manager does not display this process. The r77 service is hidden by ID by default. Uninstall.exe deletes the list of hidden process ID’s.  The “Hide”-Button can be used to write a specific process ID into the configuration system. |

The Test Console is written in C#. Following files are dependencies of the Test Console and are not part of the rootkit: BytecodeApi\*.dll, Helper32.dll, Helper64.dll, r77-x86.dll, r77-x64.dll.

## Example.exe

$77-Example.exe is useful to test task managers and file viewers. To perform a quick test on process hiding, start this executable and then use the Test Console to inject the task manager with r77. The process is no longer visible in the injected task manager. To hide the file, inject Explorer using the Test Console.



This executable does not implement anything other than a MessageBox. Any executable can be used for testing purposes by renaming its filename to start with the prefix $77.

To simulate CPU usage, change the value in the “CPU Usage” ComboBox. If this process is hidden in a task manager, the CPU usage is added to the System Idle Process. Additionally, processor usage graphs will be corrected \*.

\* Please review section 5.4 regarding issues with processor usage graphs.

## Hidden Entities

Following entities are hidden, either by prefix, by a specific condition, or by the configuration system.

“Prefix” refers to $77 at the beginning of the name (filename / registry key name / etc.)

|  |  |  |  |
| --- | --- | --- | --- |
| Entity | Hidden by prefix | Hidden by condition | Hidden by configuration |
| File | Yes |  | Hidden paths |
| Directory | Yes |  | Hidden paths |
| Named Pipe | Yes |  | Hidden paths |
| Scheduled Task | Yes |  |  |
| Process | Yes |  | Hidden PID’s  Hidden process names |
| CPU Usage |  | CPU usage of hidden processes |  |
| GPU Usage |  | GPU usage of hidden processes |  |
| Registry Key | Yes |  |  |
| Registry Value | Yes |  |  |
| Services | Yes |  | Hidden service names |
| TCP Connections |  | TCP connections of hidden processes | Hidden local TCP ports  Hidden remote TCP ports |
| UDP Connections |  | UDP connections of hidden processes | Hidden UDP ports |

### File System

On the file system, all directories and files with the prefix are hidden.



This also includes:

* File & directory junctions
* Named pipes
* Scheduled tasks (The .job file with the prefix is hidden; Scheduled tasks are hidden, when r77 is injected in the service that enumerates scheduled tasks, not mmc.exe)

In addition, individual files, directories and named pipes can be hidden by the configuration system. For this, the full path needs to be written to the configuration system:



### Processes

Processes of executables where the filename starts with the prefix are hidden.



In addition, individual process ID’s can be written to the configuration system to hide processes by ID. For files on the disk, the preferred way is to hide both the process and the executable file by prefix. For processes created in-memory, where the filename cannot be changed, hiding the process by ID is one of two options.

Alternatively, processes can also be hidden by a specific name using the configuration system:



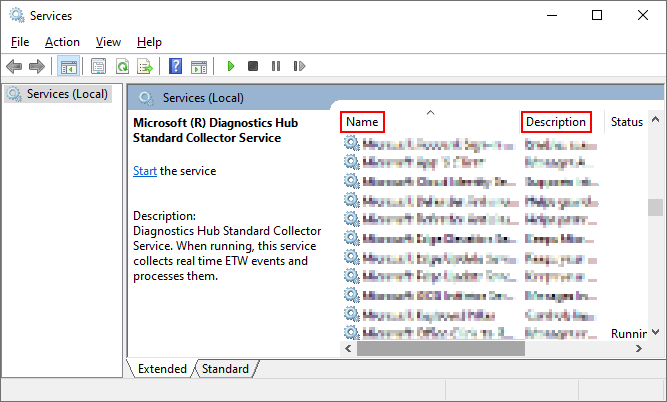
### Registry

Registry keys and values are hidden by prefix.



### Services

Services are hidden by prefix and by names specified in the configuration system. Both the name and the display name is checked against that list.



### TCP & UDP Connections

TCP and UDP connections are hidden based on either of the following:

A specific condition:

* The process is hidden by prefix.
* The process is hidden by ID.
* The process is hidden by name.

Or a specific configuration:

* The local or remote port of the TCP or TCPv6 connection is found in the configuration system.
* The port of the UDP or UDPv6 connection is found in the configuration system. UDP connections do not have a remote port.



To hide outgoing TCP connections, write the remote port into the configuration system. For example, hiding remote TCP port 443 hides all HTTPS connections created by e.g., web browsers.

To hide TCP listeners, write the local port into the configuration system.

### Windows

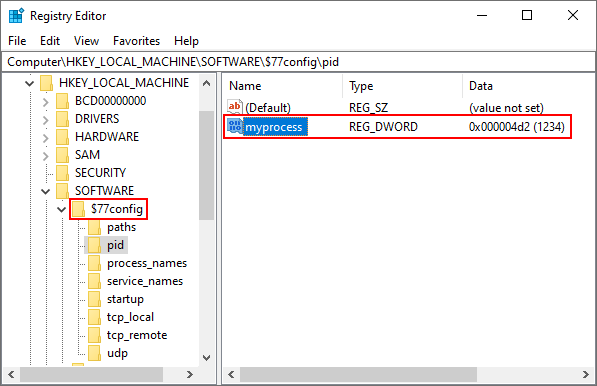
The functions EnumWindows, EnumDesktopWindows, EnumChildWindows and other functions of this group can be used to enumerate windows on the desktop. Windows of hidden processes are removed from the result set.

The desktop will still function normally, and the actual UI elements of hidden processes will remain visible.

## Configuration System

The configuration system is located in the registry under HKEY\_LOCAL\_MACHINE\SOFTWARE\$77config. The DACL of this registry key is set to allow full access to all users, even when it’s in HKLM.

Note, that when r77 is installed, the registry editor is injected with the rootkit and this registry key is not visible. Use the Test Console to detach the rootkit from regedit.



The configuration is read by r77 every 1000ms and contains following information:

* List of hidden process ID’s
* List of hidden process names
* List of hidden paths
* List of hidden service names
* List of hidden local TCP ports
* List of hidden remote TCP ports
* List of hidden UDP ports
* List of startup paths

This configuration is used to hide entities based on custom configuration in addition to the $77 prefix. Any process can write to the configuration system **without** elevated privileges.

The names of the values are generally ignored.

**Note:** Use **specific** value names – **do not** use randomized value names. When programmatically creating a new value each time, the list will grow over time and slow down the computer eventually.

### Process ID’s

The subkey $77config\pid contains DWORD values with process ID’s to be hidden. In addition, network connections from hidden processes are also hidden. This feature can be tested using the Test Console.

### Process Names

The subkey $77config\process\_names contains REG\_SZ values with filenames of processes to be hidden. In addition, network connections from hidden processes are also hidden.

**Note:** Hiding a process by ID or name instead of by prefix is only recommended, when the filename cannot have the prefix. This is particularly the case with process hollowing. The prefix also prevents the process from being injected with the rootkit.

### Paths

The subkey $77config\paths contains REG\_SZ values with full paths to files, directories, junctions, or named pipes to be hidden. Examples:

* C:\path\to\file.txt
* \\.\pipe\my\_named\_pipe

### Service Names

The subkey $77config\service\_names contains REG\_SZ values with names of services to be hidden. Both the name and the display name of services are checked against this list.

### Local TCP Ports

The subkey $77config\tcp\_local contains DWORD values with local TCP ports to be hidden.

### Remote TCP Ports

The subkey $77config\tcp\_remote contains DWORD values with remote TCP ports to be hidden.

### UDP Ports

The subkey $77config\udp contains DWORD values with UDP ports to be hidden.

### Startup Paths

The subkey $77config\startup contains REG\_SZ values with paths to files that should be executed when the r77 service starts. This occurs when Windows starts and before any user is logged on. These files (typically executables) are started under the SYSTEM account.

# Integrator’s Manual

This chapter explains:

* How to integrate r77 into another application
* How to communicate with the rootkit programmatically using the control pipe
* How to define startup files using r77

Please read section 2 before reading this chapter.

## Install.exe

Include Install.exe in your project and execute it to install r77. From this point forward, all processes are injected, and persistence is achieved. No other files are required, as they are already bundled within Install.exe.

## Install.shellcode

The file Install.shellcode is the shellcode equivalent of Install.exe.

You can simply include it in your application as a resource or BYTE[]. To invoke the file, simply

1. Load it into memory
2. Mark the buffer as RWX
3. Cast it to a function pointer
4. Execute it.

See: InstallShellCode.cpp and InstallShellCode.cs

LPBYTE shellCode = ...

DWORD oldProtect;

VirtualProtect(shellCode, shellCodeSize, PAGE\_EXECUTE\_READWRITE, &oldProtect);

((void(\*)())shellCode)();

Install.shellcode actually contains Install.exe with some shellcode at the beginning that executes the installer using process hollowing. This way, even the installer can be executed in a fileless manner.

To avoid scantime detection of the shellcode, it is recommended to encrypt the file. Do not store this file on the disk! Its purpose to make even the installation fileless would be completely defeated that way.

**Important:** Your code must be compiled in 32-bit and run with elevated privileges.

## Control Pipe

r77 provides a “control pipe”. This is a programmatic interface to communicate with the rootkit.

The control pipe is a named pipe where the r77 service receives commands from any process and executes them. This way, a process (even with low privileges) can request r77 to perform certain actions.

|  |  |  |
| --- | --- | --- |
| Control Code | Parameters | Performed Action |
| CONTROL\_R77\_TERMINATE\_SERVICE  = 0x1001 | - | Terminates the r77 service without detaching the rootkit from processes. The r77 service will restart when Windows restarts. |
| CONTROL\_R77\_UNINSTALL  = 0x1002 | - | Uninstalls r77 completely and detaches the rootkit from all processes. |
| CONTROL\_R77\_PAUSE\_INJECTION  = 0x1003 | - | Pauses injection of new processes. |
| CONTROL\_R77\_RESUME\_INJECTION  = 0x1004 | - | Resumes injection of new processes. |
| CONTROL\_PROCESSES\_INJECT  = 0x2001 | DWORD processId | Injects r77 into a specific process. |
| CONTROL\_PROCESSES\_INJECT\_ALL  = 0x2002 | - | Injects r77 into all processes. |
| CONTROL\_PROCESSES\_DETACH  = 0x2003 | DWORD processId | Detaches r77 from a specific process. |
| CONTROL\_PROCESSES\_DETACH\_ALL  = 0x2004 | - | Detaches r77 from all processes. |
| CONTROL\_USER\_SHELLEXEC  = 0x3001 | STRING file STRING commandLine | Performs ShellExecute on a specific file.  If no commandLine is required, an empty string must still be passed. |
| CONTROL\_USER\_RUNPE  = 0x3002 | STRING targetPath DWORD payloadSize BYTE[] payload | Performs RunPE. The target path must be an existing executable that matches the bitness of the payload.  The payload is an EXE file that is executed under the target path’s file. |
| CONTROL\_SYSTEM\_BSOD  = 0x4001 | - | Causes a blue screen. Can be used if required to bring the operating system to an immediate halt. |

To send a command to the r77 service, connect to the named pipe:

\\.\pipe\$77control

The first four bytes to write is the control code. Some control codes require additional parameters. These need to be written after the control code.

A STRING should be transferred as Unicode sequence of characters, followed by a two-byte null terminator.

The example in ControlPipe.cpp demonstrates how to make r77 perform a ShellExecute. The Test Console can be used to test all existing control codes.

## Enumeration vs. Direct Access

“Hiding” in r77 means that hidden entities are removed from enumerations. It is still possible to directly access a file, if the filename is known to the user – or to open a process, if the process ID is known.

Functions that open a file/process/etc. are not hooked and they do not return a “not found error” to further masquerade hidden entities. The general assumption is that the name of a hidden entity will not be guessed.

The main reason is, that there is currently no other way for r77 to maintain itself. For example, r77 could not read from the configuration system, if the hidden key is inaccessible altogether.

## Custom Startup Files

As described in section 2.6.8, it is possible to write paths to executable files into the registry. The r77 service will run these files using ShellExecute on system startup.

**The issue:** If you set up a hidden file for startup, for example using the HKCU\...\Run key, Windows cannot not find the file because it is hidden and therefore it does not start.

**The solution:** r77 is in charge of starting hidden files. This comes with several advantages:

1. Your file will start under the SYSTEM account with system integrity.
2. Your file will start before the first user is logged on.
3. You can add files to startup with non-elevated privileges and they will start up with system integrity.

If you want your process to be run under a specific user account, you have to perform impersonation. This is required in case you need access to the user’s desktop.

**Note:** Just by adding the file to $77config\startup, it is not implicitly hidden. The same rules apply: The file must have the prefix, or it must be hidden by the configuration system. If you want the file to not be injected by r77, then writing the helper signature to the executable file will avoid injection (see section 4.9).

# Implementation Details

This chapter explains implementation details in-depth that are important to understand, if

* … you want to modify the code and change or add features
* … you want to create your own FUD version of r77

This is a very advanced chapter and requires deep understanding of certain concepts that are used throughout the project, such as hooking, in-memory techniques, the PE format, etc.

## Compilation

r77 can be compiled in just one step by building the solution. The compiled output will be written to the $Build\ directory, so that all relevant files are in one place.

**Important:** Compile with **Release** and not Debug configuration! Injecting a debug-build DLL into remote processes will not work correctly.

The BuildTask.exe project is used in Post-Build events to help with compilation.

## Rootkit DLL

The r77 Rootkit is a DLL file (r77-x86.dll and r77-x64.dll) that is compiled separately for 32-bit and 64-bit processes. Once injected into a process, this process will not show hidden entities.

r77 implements reflective DLL injection. The DLL does not need to be written to the disk at any time. Instead, the file is written to the remote process memory and the ReflectiveDllMain export is called to finally load the DLL and invoke DllMain. For this reason, the DLL is not listed in the PEB.

Injecting the DLL into a process that is already injected has no implications. DllMain will detect this and just return FALSE to unload itself.

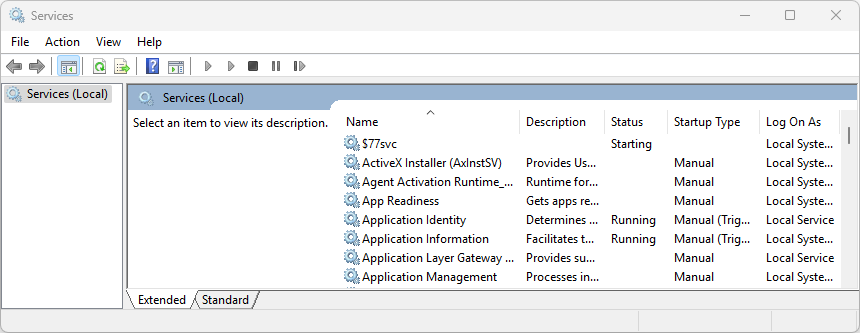
## r77 Service

When Install.exe is executed, the r77 service is set up and started. The r77 service is fileless, which means the installer does not write any files to the disk.

The primary purpose of the r77 service is to inject all running processes when it starts, as well as injecting processes that are created later on.

### Fileless Startup

**Stage 1:** The installer creates a Windows service, named $77svc.

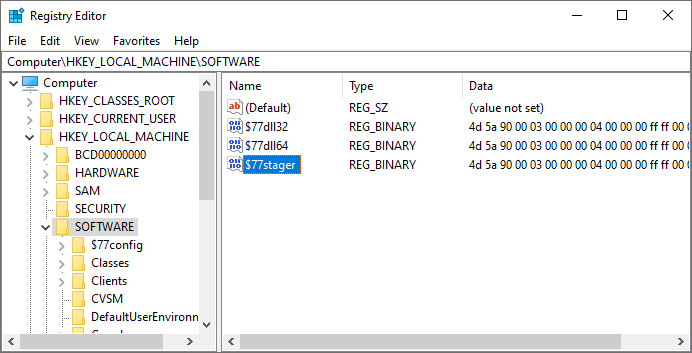


This Windows service does **not** start the r77 service executable from disk. Instead, it starts powershell.exe with following command line:

[Reflection.Assembly]::Load([Microsoft.Win32.Registry]::LocalMachine.OpenSubkey('SOFTWARE').GetValue('$77stager')).EntryPoint.Invoke($Null,$Null)

**Note:** In version 1.6.x, a scheduled task was used instead of a Windows service.

The command is inline and does not require a .ps1 script. Here, the .NET Framework capabilities of PowerShell are utilized to load a C# executable from the registry and execute it in memory. For this, Assembly.Load().EntryPoint.Invoke() is used.



The r77 startup is considered **fileless Type II,** because the above commandline does reside in the registry and is located on the disk in theory. However, the registry is a proprietary container, whereas the actual filesystem is not touched during initialization.

In addition, the inline script must bypass AMSI to evade AV detection (see section 4.8.1).

**Stage 2:** The executed C# binary is the stager. It will start the r77 service by injecting it into the winlogon process using reflective DLL injection.

**Note:** In version 1.5.x, process hollowing was used to launch the r77 service. Starting from version 1.6.0, the r77 service does not have its own process.

Since the Windows service starts PowerShell under the SYSTEM account, the r77 service also runs under the SYSTEM account. Therefore, it can inject processes with system IL, except for protected processes, such as services.exe.

**Important:** No EXE or DLL files are written to the file system. Even dropping Install.exe can be avoided by using the shellcode installer to deploy r77 in a completely fileless manner.

**Stage 3:** The r77 service process is now running. Following operations are performed:

1. All running processes are injected.
2. A named pipe is created to handle injection of newly created child processes.
3. In addition to child process hooking, a subroutine checks for newly created processes every 100ms. This is because some processes cannot be injected, but still create child processes. This is particularly the case for services.exe, which is a protected process.
4. The control pipe is created, which handles requests (commands) received by other processes.
5. Files under $77config\startup are executed.

## Child Process Hooking

When the r77 service starts, all currently running processes are injected. Processes that spawn later must be injected, too. There are two concepts to achieve this:

**1.) Hooking the creation of processes:** A function that is always called when a process is created is NtResumeThread. When process A creates a process B, this function is called by process A after process B is fully initialized, but still suspended. After this function call, the creation of process B is completed.

Therefore, NtResumeThread is hooked and r77 is injected into the new process **before** actually calling this function. Unfortunately, 32-bit processes can spawn 64-bit child processes. Since a 32-bit process cannot inject a 64-bit process, the r77 service provides a named pipe to handle injection requests. When sending the new process ID to the r77 service, the service injects the process and returns a confirmation. After receiving the confirmation from the service, the injection has completed and NtResumeThread can be called.

This way, a process is injected with r77 before it can execute any of its own instructions. This is very important, because some programs (e.g., RegEdit) initialize fast, then perform enumerations and display the result shortly after starting. The rootkit must be injected at the very beginning!

However, unfortunately, not all processes can be injected. Windows 10 protects certain processes from access, such as services.exe. So, relying on child process hooking alone would result in services spawning without r77 injected, among other processes. This is when **concept 2** comes into play.

**2.) Hooking new processes periodically:** Every 100ms, a list of running processes is retrieved. Any new process in that list will be injected. This way, processes that are missed by the child process routine are still injected. However, there is a delay of up to 100ms, where r77 is not running in that process.

As mentioned previously, double injection of a process has no negative impact. It is supported behavior.

## Shellcode Installation

Install.shellcode is the shellcode equivalent of Install.exe.

This file starts directly with executable code that loads Install.exe, which is stored at the end of the shellcode. The shellcode is written in assembler and simply performs process hollowing to run the installer executable.

This saves you the time, writing your own RunPE.

The compiled shellcode is less than 1 KB and both files are simply concatenated upon compilation, thus generating Install.shellcode when the solution is built.

## Dependencies & Requirements

r77 does not have any dependencies, other than the operating system itself and the tools that are already present after the initial OS installation. The binaries are written in C and do not require the Visual C++ runtime to be installed.

However, the fileless startup mechanism requires PowerShell and .NET Framework. Both dependencies are present on a clean installation of Windows 7, 10 and 11.

The .NET Framework normally has the issue that .NET 2.0-3.5 and .NET 4.0-4.8 are two distinct CLR’s. This means that .NET executables targeting .NET 3.5 do not run, when only .NET 4.x is installed – and .NET executables targeting .NET 4.x do not run, when only .NET 3.5 is installed. However, this is **not an issue** for the r77 stager.

On Windows 7, .NET 3.5 is installed by default and on Windows 10, .NET 3.5 is not installed, but 4.x instead. When executing a C# binary in memory from PowerShell as described in section 4.3.1, the target version is irrelevant. The target framework of the fileless stager is set to .NET 3.5 to avoid any code that is incompatible with .NET 4.x. However, the stager will run, if either .NET 3.5 *or* .NET 4.x is installed.

Therefore, this requirement is always met. r77 is **not** a “.NET rootkit”, because only the startup code requires .NET, but the rootkit itself is written in C completely. There is no practical way to achieve fileless persistence without using Powershell or .NET.

### Elevated Privileges

The full installation with persistence requires elevated privileges. Escalating privileges using an exploit or a UAC bypass technique is not part of this project.

When using the Test Console running with medium IL, the r77 DLL can be injected into processes with medium IL, but not elevated ones. This is practical enough to do some tests, however a full installation makes no sense, when elevated processes are not injected.

## Hooked API’s

Detours is the hooking library used to hook functions from ntdll.dll. This DLL is loaded into every process on the operating system. It is a wrapper around all syscalls, which makes it the lowest layer available in ring 3. Any WinAPI function from kernel32.dll or other libraries and frameworks will ultimately call ntdll.dll functions. It is not possible to hook syscalls directly. This is a common limitation to ring 3 rootkits.

Hiding of services exceptionally requires hooking of advapi32.dll and sechost.dll instead. Please read section 4.7.10 about why this is a requirement.

The following chapters describe each function that is hooked.

### NtQuerySystemInformation

This function is used to enumerate running processes and to retrieve CPU usage.

### NtResumeThread

This function is hooked to inject created child processes, while the new process is still in a suspended state. Only after injection has completed, this function is actually called.

**Note:** Hooking the CreateProcess API is not a good alternative, because it creates and starts a process in one go. Also, it is a high-level API, of which there are several. It would be bad design to hook numerous functions that are similar, only to achieve one task.

### NtQueryDirectoryFile

This function enumerates files, directories, junctions and named pipes.

### NtQueryDirectoryFileEx

This function is very similar to NtQueryDirectoryFile and it must be hooked as well. The implementation is mostly the same.

The dir command in cmd.exe uses this function instead of NtQueryDirectoryFile.

### NtQueryKey

This function is called by RegQueryInfoKey and is primarily used to retrieve the number of subkeys and values. This number needs to be subtracted by the number of hidden subkeys and values in order to match successive calls to NtEnumerateKey and NtEnumerateValueKey.

### NtEnumerateKey

This function is used to enumerate registry keys. The caller specifies the index of a key to retrieve it. To hide registry keys, the index must be corrected. Therefore, the key must be enumerated again to find the correct “new” index.

### NtEnumerateValueKey

This function is used to enumerate registry values. The implementation of this hook is very similar to NtEnumerateKey.

### NtUserBuildHwndList

This function is called by EnumWindows, EnumDesktopWindows, EnumChildWindows, etc. Any time when enumerating windows, those of hidden processes are excluded. They are still visible on the desktop and functioning normally.

### EnumServiceGroupW

This function is used to enumerate services.

### EnumServicesStatusExW

This function is similar to EnumServiceGroupW.

**Note:** Both functions communicate with services.exe through RPC to retrieve a list of services. A hook in ntdll.dll would have no effect, because only services.exe uses the ntdll functions. Therefore, the higher-level DLL’s advapi32.dll and sechost.dll are hooked.

**Note:** For service hiding, only the Unicode functions are hooked, because EnumServicesStatusExA, etc. do not seem to be used by any applications out there. There is just a lack of real-world application to actually test the ANSI analogues on.

### NtDeviceIoControlFile

This function is used to access drivers using IOCTL’s.

If the driver is \Device\Nsi and the IOCTL is 0x12001b, a list of all TCP and UDP connections is requested by the caller.

To hide a row, all following rows need to move up by one and the total count needs to be decreased.

### PdhGetRawCounterArrayW

This API is used to retrieve GPU usage. In the filtered array, szName is a string like pid\_1234\_luid\_0x00000000\_0x0000C9DE\_phys\_0\_eng\_0\_engtype\_3D, from which a process ID can be deducted. If this process is hidden, the performance counter values are set to 0.

**Note:** This could be handled at a lower level in NtDeviceIoControlFile by filtering output from the \Device\PcwDrv device and the 0x224013 IOCTL. So far, the exact format of the data that this driver returns is unknown. A later version of r77 might hook the IOCTL instead of pdh.dll.

### PdhGetFormattedCounterArrayW

Same implementation as in PdhGetRawCounterArrayW.

### AmsiScanBuffer

This API is called, when Powershell, the .NET Framework, or any other AMSI supporting application sends a buffer to AV for analysis. Hooking this function disables AMSI systemwide.

## AV Evasion Techniques

Several techniques to evade AV/EDR detections have been implemented.

### AMSI Bypass

The stager is started by Powershell. The Powershell script uses Assembly.Load().EntryPoint.Invoke() to load the C# stager executable from the registry and invoke it. However, AMSI is implemented in both Powershell and the .NET Framework itself. A call to Assembly.Load() will trigger AMSI and send the executable to AV for analysis. To bypass this, AMSI must be disabled for the entire Powershell process.

The function amsi.dll!AmsiScanBuffer must be patched to always return AMSI\_RESULT\_CLEAN. In Install.c, the Powershell startup script is composed to contain code that performs this patch. Powershell variable names are dynamically obfuscated each time r77 is installed. In addition, string literals are obfuscated in a polymorphic fashion to evade signatures of strings, such as 'amsi.dll'.

**Note:** The Powershell code must not contain any Add-Type cmdlets with C# code in them. It would invoke csc.exe (.NET compiler), which would drop a C# DLL to the disk. Instead, a return-to-libc-like approach is used by using reflection to find certain .NET functions.

**Note:** Bypassing AMSI is only required on Windows 10 and 11, as AMSI is not implemented in Windows 7.

In addition to bypassing AMSI during startup, which is really required, there is also a hook on the AmsiScanBuffer function. This disables AMSI systemwide after the rootkit is running.

### DLL Unhooking

Many EDR solutions implement hooks in ntdll.dll, and often in kernel32.dll and kernelbase.dll. Those hooks monitor API calls, particularly those required for code injection, process hollowing, etc. To evade detection of process hollowing, EDR hooks need to be removed.

Unhooking must be performed in:

* **Stager:** The C# executable that is invoked by Powershell creates the r77 service by injecting it into the winlogon process. To evade detection of code injection, EDR hooks need to be removed.
* **r77 service:** Because the r77 service injects all running processes, EDR hooks must be removed here as well.

EDR hooks are typically a jmp instruction at the beginning of several suspicious ntdll functions. Those hooks are trivial to remove because they exist in user mode only. EDR does not typically implement kernel mode hooks.

Removing EDR hooks is achieved by loading a fresh copy of ntdll.dll from disk and replacing the currently loaded .text section of the ntdll module with the original unhooked file contents.

However, the EDR might flag file reads to ntdll.dll, therefore syscalls must be used instead of NtCreateFile to read ntdll.dll. EDR will also statically flag the syscall instruction, therefore we must find a syscall ret gadget and jump to it. The syscall numbers of the required functions can be deduced by sorting exports of ntdll by their case insensitive name. This is needed, because the mov eax gadget that tells us the syscall number is most likely destroyed by the EDR.

## r77 Header

To mark a process as injected, or as the r77 service, etc., the “r77 header” is used.

The most suitable location in the memory of a process to write the r77 header to is the DOS stub. Because it is not used for anything, the process will not malfunction if bytes are overwritten here:



Note, that this is the view of an executable’s memory and **not** the file on the disk.

A process may contain either of the following header. The signature is written to the first two bytes of the DOS stub. Any following data is written after the signature.

|  |  |
| --- | --- |
| Signature & default value | Description |
| R77\_SIGNATURE  = 0x7277 | When the r77 DLL is injected, R77\_SIGNATURE is written to the main module’s DOS stub.  This way, the Test Console can detect, whether or not a process is injected.  If r77 is injected into a process, but the r77 signature is already present, DllMain returns FALSE to avoid double injection.  R77\_SIGNATURE is followed by a function pointer to DetachRootkit(). Calling it will detach r77 from the process gracefully. This is used by the Test Console and Uninstall.exe.  When detaching r77 from a process, the r77 header is removed by restoring the DOS stub to its original state. |
| R77\_SERVICE\_SIGNATURE  = 0x7273 | The process that hosts the r77 service (winlogon.exe) needs to be identifiable to Install.exe and Uninstall.exe.  If r77 is injected into the r77 service process, DllMain also returns FALSE. |
| R77\_HELPER\_SIGNATURE  = 0x7268 | The helper signature is written to helper files at compile time. These include any executables of the test environment, such as TestConsole.exe.  If r77 is injected into a helper process, DllMain also returns FALSE.  You can write the helper signature to your own files at **compile time**, if you want r77 to not be injected into them. |

## Compile Time Constants

Compile time constants are defined in both r77def.h and R77Const.cs. Changes must be applied in both files.

The r77 signatures are already mentioned in section 4.9. When changing these, r77 can be customized to be distinct from and be undetectable by the publicly available r77 binaries.

This is a list of additional constants that can be modified:

|  |  |
| --- | --- |
| Constant & default value | Description |
| HIDE\_PREFIX  = "$77" | The prefix by which entities are hidden. |
| R77\_SERVICE\_NAME  = HIDE\_PREFIX + "svc" | The name of the Windows service that launches the r77 service process. |
| CHILD\_PROCESS\_PIPE\_NAME  = "\\\\.\\pipe\\" + HIDE\_PREFIX + "childproc" | The name of the named pipe that is used for child process hooking requests. |
| CONTROL\_PIPE\_NAME  = "\\\\.\\pipe\\" + HIDE\_PREFIX + "control" | The name of the control pipe as described in section 3.3. |
| PROCESS\_EXCLUSIONS  = { "MsMpEng.exe", "MSBuild.exe" } | Hardcoded list of processes that will not be injected. |

## De-implementation of features

It might be desirable to de-implement certain features, if…

* … a feature is not needed
* … a feature could increase the likelihood of detection
* … you want to reduce executable size

Fortunately, the code of this project is carefully maintained and refactored regularly. Here are some examples of how to de-implement certain features to produce a tailored build of r77:

**Disable a hooked function**

In Hooks.c, edit the functions InitializeHooks and UninitializeHooks. Remove lines with InstallHook and UninstallHook to de-implement certain hooks. Make sure to read the comments in the hooked function to understand what it is that you’re removing. Some hooks might be interdependent. Do not de-implement NtResumeThread, as it’s a core component and required for child process hooking.

This can be useful, if you only intent to hide certain entities (e.g., you want to hide processes, but you don’t care about hiding files).

**AMSI Bypass**

In Install.c in the function GetPowershellCommand, remove the block if (IsAtLeastWindows10()) where the AMSI bypass is appended to the powershell command. The command should start directly with L"[Reflection.Assembly]::Load(....

**Disable the control pipe**

In Service.c, remove the line ControlPipeListener(ControlCallback);. The r77 service should then compile without the control pipe feature.

… These are just a couple of examples, but removing any other feature should be simple and involve just single lines of code to be removed.

# Notes

## Supported Platforms

Windows 11, Windows 10 and Windows 7 are supported, including both x64 and x86 editions. All throughout the product, operating system bitness is taken into consideration. When the documentation mentions x64 and x86 distinction, this only applies to the 64-bit edition. On 32-bit Windows, only 32-bit components are installed.

Supported operating systems are based on market share rather than official support by Microsoft.

|  |  |  |  |
| --- | --- | --- | --- |
| Operating System | x64 | x86 | Market share \* |
| Windows 11 | Supported | Supported | 34 % |
| Windows 10 | Supported | Supported | 63 % |
| Windows 7 | Supported | Supported | 2,4 % |
| Windows 8.1 | Not supported | Not supported | < 1 % |
| Windows 8 | Not supported | Not supported | < 1 % |

\* Statistics are taken from Statcounter in January 2025.

r77 is tested on all supported operating systems prior to release.

## Compatibility

Rootkits, in general, are designed to work for any program, not just specific applications, like Explorer.exe and TaskMgr.exe. r77 hooks functions in ntdll.dll, which is the lowest layer available in ring 3. Therefore, any program is compatible with r77, including programs that will be developed in the future.

## Tested Applications

There is a set of applications that are used to test each module. However, r77 should work for any other application equally.

Applications used in testing:

* Windows Task Manager \*
* Process Explorer
* Process Hacker
* Windows Explorer
* Windows Registry Editor
* Services.msc
* TCPView
* CurrPorts
* cmd.exe (dir, netstat.exe)

\* See section 5.4 regarding known issues with the listed applications.

To report bugs regarding applications that behave incorrectly, regardless of whether or not they are in the list of tested applications, please read section 5.6.

## Known Issues

To ensure quality and compatibility, r77 is tested with several operating system versions and well-known applications. This is a list of all issues that are known and need to be resolved in future releases.

|  |
| --- |
| Issues |
| Sandboxed processes cannot be injected. A sandbox is defined by an integrity level of LOW or UNTRUSTED. Typical applications are sandboxed processes by web browsers and document readers. Injecting a sandboxed process is causing crashes of the process and therefore, r77 does currently not inject into sandboxes. |
| Injecting critical processes, such as smss.exe, csrss.exe, or wininit.exe, has been reportedly causing issues. The exact reason is unknown, therefore processes are only injected, if they are not marked as critical. |
| Some specific executables are not working correctly when r77 is injected. MSBuild.exe is one example and therefore on the exclusion list. |
| Since Windows 8, processes can be in a suspended state. This is particularly relevant for Windows apps. A Windows app is suspended once the main window is minimized.  Injecting those processes result in odd behavior, such as injection does not complete before the process is resumed. Also, detaching such processes does not work while the process is in a suspended state. When uninstalling r77, suspended processes are not detached, and a reboot may be necessary to clean running Windows app processes from the injected DLL. This needs to be handled properly. However, this is a minor issue that does not cause instability or malfunction. |
| Hiding CPU usage only works partially. The CPU usage of hidden processes is accumulated to the System Idle Process correctly.  However, the graphs of task managers that display CPU usage either remain unchanged, or do not display the correct values.  In detail:   * TaskMgr and perfmon graphs are not corrected, at all. * ProcessHacker: The graph that shows individual CPU cores has spikes in it, because the algorithm currently assumes, that CPU usage is equally distributed across all logical processors. It needs to be calculated per process AND per core at the same time, which is not implemented, yet.   Use $77-Example.exe to test CPU usage hiding. |

## ToDo List

Following features are on the agenda for upcoming releases:

* Hide loaded DLL’s that have the prefix from the PEB of any process.
* Prohibit querying/opening/deleting/etc. operations on hidden entities (see section 3.4).
* More features for the control pipe:
  + Execute shellcode
  + Inject DLL into any process

## Bug Reports

Please feel free to report any bugs that are not in the list of known issues. Either create an issue in the [GitHub](https://github.com/bytecode77/r77-rootkit) repository or visit [bytecode77.com/contact](https://bytecode77.com/contact) to get in touch.

Bug reports are only considered for supported operating systems (See section 5.1).

**In scope**

* Processes crash or behave abnormally, when r77 is injected.
* Processes cannot be injected for reasons, not stated in the documentation.
* Hidden entities are visible, even though r77 is injected. Any application is in scope, not just the list of explicitly tested applications.
* The r77 service fails to start at system startup.
* The r77 service crashes.
* Installation fails.
* Uninstallation fails or does not remove r77 completely.
* A bug in the Test Console or the test environment in general.
* Anything that is not explicitly mentioned above but constitutes as a bug.

**Out of scope**

* Memory regions that are not zeroed-out. Example: An enumeration is hooked, one item is removed and the count is decreased. Anything that is out of bounds of the new array is not zeroed-out, because it is expected that the caller does not read beyond the buffer to find hidden entities.
* Notoriously crafted parameters of hooked functions that cause the r77 DLL to crash or malfunction in a way that would not happen with normal usage. Penetration and fuzzing of the DLL is not in scope. However, coding errors like missing NULL checks on parameters are in scope.
* Revealing the rootkit by using debuggers is out of scope. Using kernel mode debugging is out of scope completely. r77 is hiding entities from normal to intermediate computer users, not from pentesters.
* Kernel code is out of scope, because r77 is a ring 3 rootkit. It does not hide anything from the kernel or from kernel mode drivers.
* Detection by specific AV vendors; r77 is designed to be very evasive. The fileless concept is the cornerstone that makes it even possible. However, r77 is an open-source project and AV vendors will eventually detect the rootkit. Implementing evasion for a specific AV vendor is a daunting task and once the AV has updated detection routines, the rootkit will eventually be detected again. If you want it to be FUD, then you have to do the modifications yourself. The only case where evasion is impossible to achieve is, if r77 would reside on the disk, which is not the case.
* Scan-time detection of Install.exe. Use the shellcode installer or execute this file using RunPE.

# Change Log

|  |  |  |
| --- | --- | --- |
| Version | Release | Changes |
| 1.7.1 | 27.08.2025 | * Hook NtUserBuildHwndList to hide windows of hidden processes. |
| 1.7.0 | 09.03.2025 | * Install using Windows service instead of scheduled task. * NtResumeThread calls the named pipe only for x86 -> x64 injections and otherwise injects directly. |
| 1.6.3 | 03.02.2025 | * Hook NtQueryKey. |
| 1.6.2 | 24.01.2025 | * Bugfix: Hang in NtResumeThread causes stability issues. |
| 1.6.1 | 09.01.2025 | * Bugfix: Unhooking EDR in every process causes stability issues. |
| 1.6.0 | 08.01.2025 | * The r77 service is injected into winlogon, instead of having its own process created using process hollowing. |
| 1.5.5 | 01.01.2025 | * Unhook ntdll using indirect syscalls. * Obfuscate the shellcode that overwrites AmsiScanBuffer. * Bugfix: Poor performance of NtEnumerateKey and NtEnumerateValueKey leads to crashes, e.g. in EventVwr. * Bugfix: Linking pdh.lib causes hangs in Windows 7. |
| 1.5.4 | 13.12.2024 | * Bugfix: GetExtendedTcpTable ignored hidden process names. |
| 1.5.3 | 02.12.2024 | * Bugfix: HookedNtQueryDirectoryFile ignored the returnSingleEntry parameter. |
| 1.5.2 | 02.05.2024 | * Hook AmsiScanBuffer to disable AMSI systemwide, in every process. |
| 1.5.1 | 30.11.2023 | * GPU usage of hidden processes is hidden. |
| 1.5.0 | 02.09.2023 | * Only one r77 service process is started. Previously, two were required to handle both 32-bit and 64-bit processes. Full compatibility is maintained when upgrading. * Process hollowing routines set memory protections correctly. There no longer are any RWX sections. * TestConsole helpers are now DLL’s instead of EXE files. |
| 1.4.3 | 06.06.2023 | * Bugfix: Installation failed when notebook was not plugged into power. |
| 1.4.2 | 22.10.2022 | * Polymorphic AMSI bypass code (evades Windows Defender signatures of Powershell strings) |
| 1.4.1 | 21.10.2022 | * Written in plain C (previously C++) * Reduced file size by 50%. * Bugfix: Crash in TcpView.exe and similar applications due to bug in NtDeviceIoControlFile hook. * Config system editable in Test Console. |
| 1.4.0 | 01.09.2022 | * Shellcode installation (see section 3.2). |
| 1.3.0 | 01.05.2022 | * PROCESS\_EXCLUSIONS: Hardcoded list of processes that will not be injected. * Control pipe: Programmatic interface over which r77 receives commands from other processes (see section 3.3). |
| 1.2.2 | 31.08.2021 | * Custom Startup Files (see section 3.5) to address the issue that Windows does not consider hidden files in startup, because Windows cannot “see” these files. |
| 1.2.1 | 20.06.2021 | Several AV evasion techniques have been implemented (see section 4.8).   * Bypass AMSI detection of the Powershell startup by overwriting amsi.dll!AmsiScanBuffer with a ret. * Unhook ntdll.dll and kernel32.dll in both the stager and the r77 service to evade EDR detection. * Hook sechost.dll instead of api-ms-\*.dll. |
| 1.2.0 | 18.04.2021 | * Hide services by prefix and name. * Breaking Change: The configuration system is now under HKEY\_LOCAL\_MACHINE\SOFTWARE\$77config exclusively. The DACL is set to allow full access to all users. The key HKEY\_CURRENT\_USER\Software\$77config is no longer considered. This architectural change was made, because some processes (e.g., those under the NETWORK SERVICE account) have no read access to the HKU registry hive. * The helper files of TestConsole.exe are combined into one set of files: Helper32.exe and Helper64.exe. |
| 1.1.0 | 11.04.2021 | * Hide processes by name * Hide files, directories, junctions and named pipes by full path |
| 1.0.1 | 05.03.2021 | * Bugfix: Crash when injecting critical processes (e.g., smss.exe, csrss.exe, or wininit.exe). Resolution: Do not inject critical processes. |
| 1.0.0 | 21.02.2021 | **Initial release**   * Full rewrite * Resolved all issues of the beta release * Uses Detours instead of MinHook * Implements proper & out of the box installation & persistence * The rootkit is fileless * Testing framework * … And a lot more |
| 0.6.0 | 17.12.2017 | **Beta release** |