r77 Rootkit

Technical Documentation

r77 Version	1.3.0
Release date	01.05.2022

Author	bytecode77
Website	bytecode77.com/r77-rootkit
GitHub	github.com/bytecode77/r77-rootkit

Table of Contents

1 Introduction

r77 Rootkit is a fileless ring 3 rootkit. Its primary purpose is to hide certain entities, such as files, directories, processes, etc.

Additionally, the rootkit comes with an out of the box installer that handles injection of processes and persistence. The installation is completely fileless, meaning no files need to be written to the disk. r77 solely relies on in-memory operations and remains on the system after reboot.

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

This documentation is targeted to integrators of r77 and developers who aim to modify the code of r77.

1.1 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	?
Windows 10	Supported	Supported	58 %
Windows 7	Supported	Supported	25 %
Windows 8.1	Not supported	Not supported	3 %
Windows 8	Not supported	Not supported	< 1 %

* The market share statistics are taken from netmarketshare on May 2022.

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

1.2 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.

1.3 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 6 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, go to section 8.

1.4 Dependencies & Requirements

r77 does not have any dependencies, other than the operating system itself and the tools that are already present after the initial installation. The binaries are written in C++ and compiled with /MT.

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

The .NET Framework normally has the issue, where .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 2.4.1, 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.

1.4.1 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.

2 Rootkit

2.1 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.

2.2 Installer

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. r77 is set up to start after reboot and inject all processes before the first user is logged on.

`Install.exe` has both `r77-x86.dll` and `r77-x64.dll` included in its PE resources. It is not necessary to deploy the DLL's along with it. This is a **single file deployment**. `Install.exe` can also be executed using process hollowing to avoid writing the installer to the disk during deployment.

When executing `Install.exe` a second time after r77 is already installed, the r77 service processes are terminated and recreated. 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, use `Uninstall.exe`.

Please review section 5 for details on how to integrate the installer into your own project.

2.3 Uninstaller

To remove r77 from the system completely, run `Uninstall.exe`. It will uninstall r77 in following steps:

1. Delete the `$77stager` value from the registry.
2. Delete the scheduled task.
3. Terminate the r77 service.
4. Detach r77 DLL from all injected processes.
5. If the operating system is a 64-bit operating system, all above steps need to be performed again, but from within a 64-bit executable. For this, an executable with a random filename is dropped in the temp directory, executed, and deleted afterwards. This executable is embedded in the PE resources of `Uninstall.exe`.
6. Delete the `$77config` key from the registry.

Executing `Uninstall.exe` a second time has no effect. However, if any of the above steps failed, it would clean up remaining leftovers.

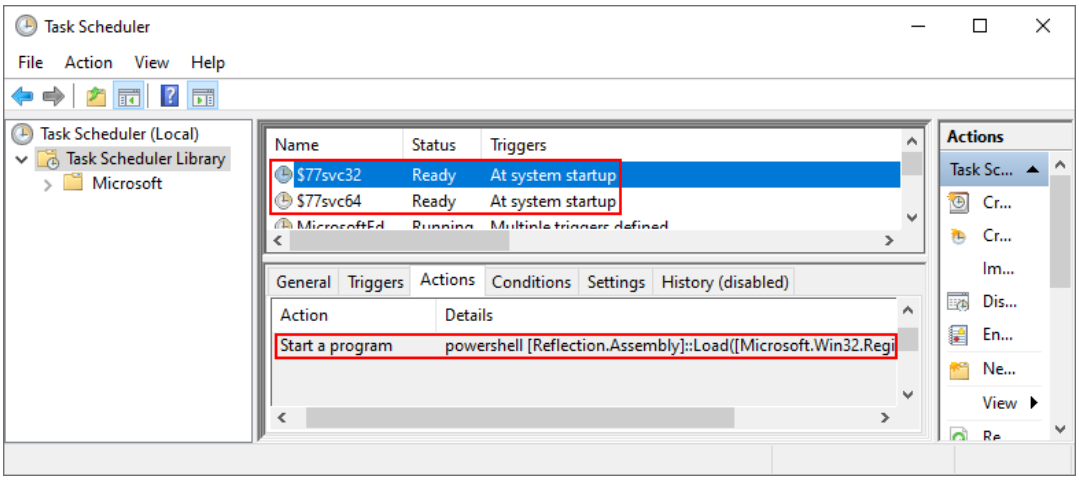
2.4 r77 Service

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.

Two separate r77 service processes are needed to inject both 32-bit and 64-bit processes. The primary purpose of the r77 service is to inject all running processes when the r77 service started, as well as injecting processes that are created later on.

2.4.1 Fileless Startup

Stage 1: The installer creates two scheduled tasks for both the 32-bit and the 64-bit r77 service. A scheduled task does require a file, named `$77svc32.job` and `$77svc64.job` to be stored, which is the only exception to the fileless concept. However, scheduled tasks are also hidden by prefix once the rootkit is running.

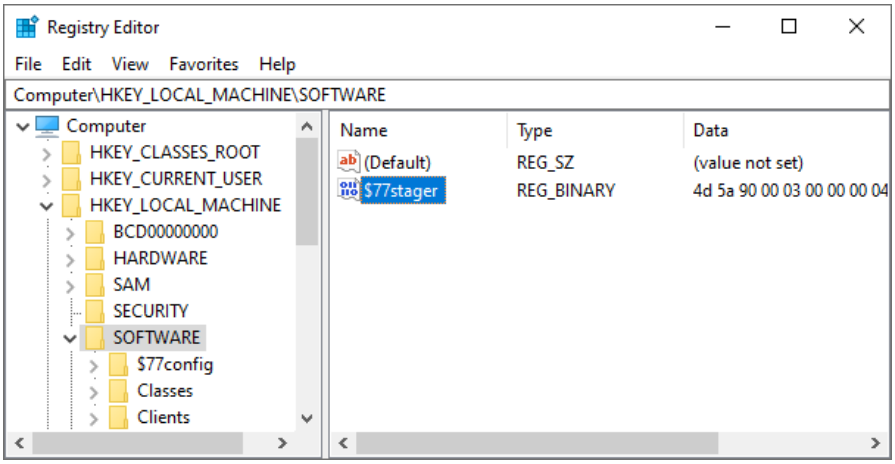


The scheduled task does **not** start the r77 service executable from disk. Instead, it starts `powershell.exe` at system startup with following command line:

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

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

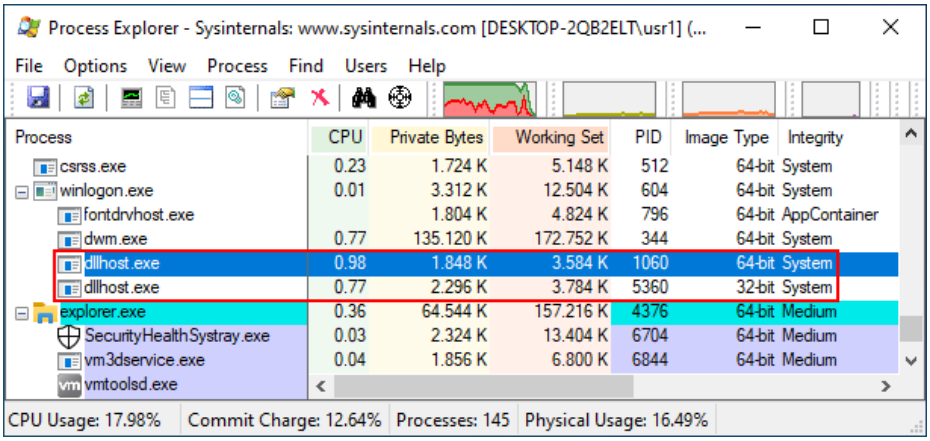
In addition, the inline script must bypass AMSI to evade AV detection (see section 4.5.1).



Stage 2: The executed C# binary is the stager. It will create the r77 service processes using process hollowing.

The r77 service is a native executable compiled in both 32-bit and 64-bit separately. The parent process is spoofed and set to `winlogon.exe` for additional obscurity. In addition, the two processes are hidden by ID and not visible in the task manager.

Since the scheduled task 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: The only items written to the file system are the job files (`$77svc32.job` and `$77svc64.job`) and the registry value `$77stager` with the stager executable. No EXE or DLL files are stored in the file system directly. Even `Install.exe` can be executed using process hollowing to deploy r77 in a completely fileless manner. This is very important, because the rootkit installer or the included DLL files may be detected by AV and deleted.

Stage 3: Both r77 service processes are now running. Following operations are performed:

1. The process ID is stored in the configuration system to hide the processes. Because the processes were created using process hollowing, they cannot have the `$77` prefix.
2. All running processes are injected.
3. A named pipe is created to handle injection of newly created child processes.
4. 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.

5. The control pipe is created, which handles requests (commands) sent by other processes to the rootkit.
6. Files under `$77config\startup` are executed.

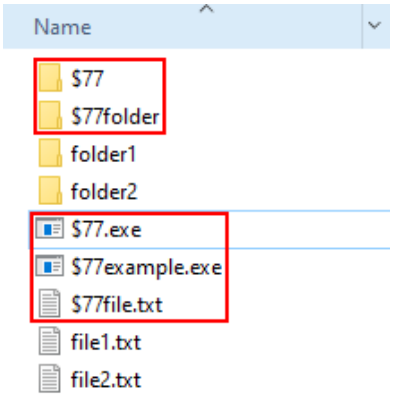
2.5 Hidden Entities

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

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	
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

2.5.1 File System

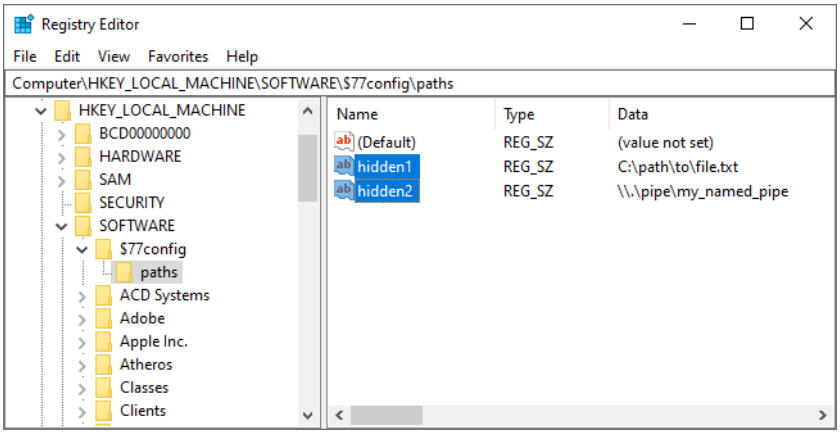
On the file system, all directories and files with the prefix are hidden.



This also includes:

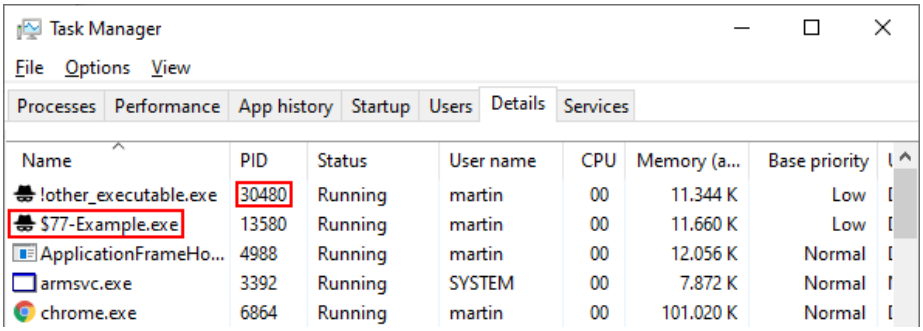
- File & directory junctions
- Named pipes
- Scheduled tasks (`.job` file with the prefix are hidden; Scheduled tasks are hidden, when r77 is injected in the service that enumerates scheduled tasks, not `mmc.exe`.)

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:



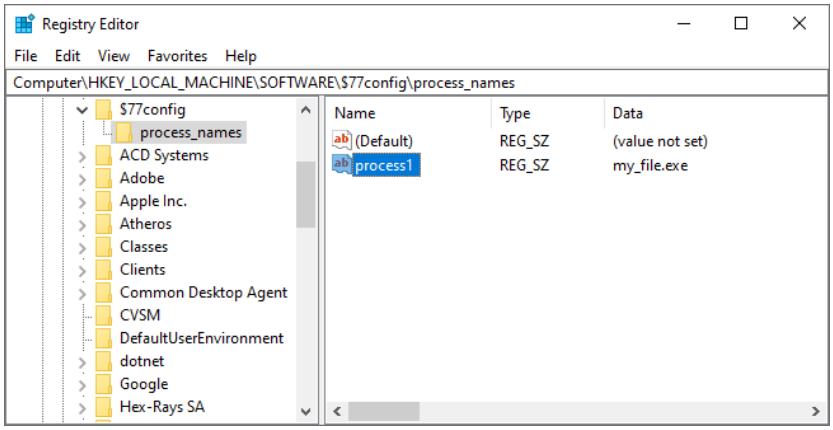
2.5.2 Processes

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



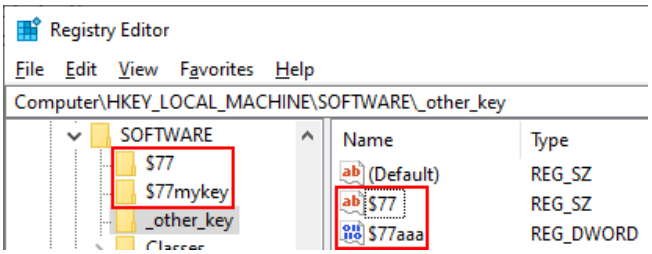
In addition, individual process ID's can be written to the configuration system. These process ID's are picked up by r77 and in effect, processes are hidden by ID. For files on the disk, the preferred way is to hide both the process and the executable file by prefix. For processes created in-memory, 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:



2.5.3 Registry

Registry keys and values are hidden by prefix.



2.5.4 TCP & UDP Connections

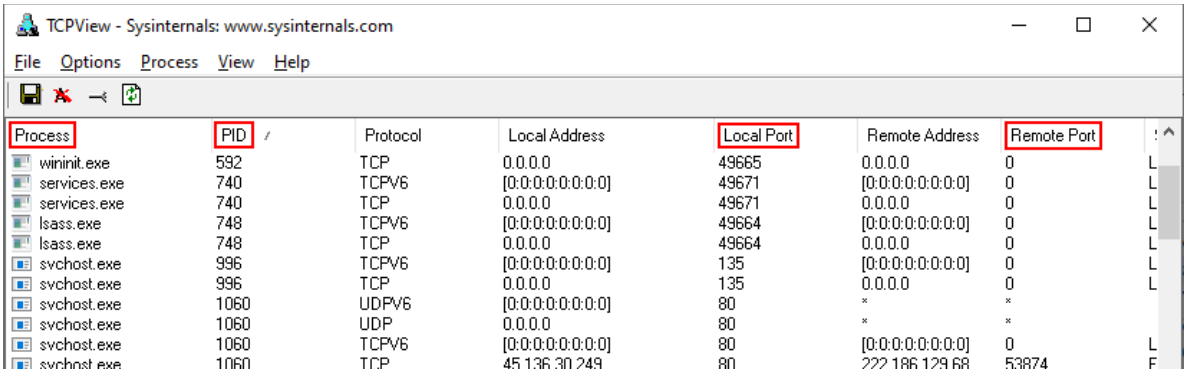
TCP and UDP connections are hidden based on either of the following:

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.

2.6 Hide Prefix

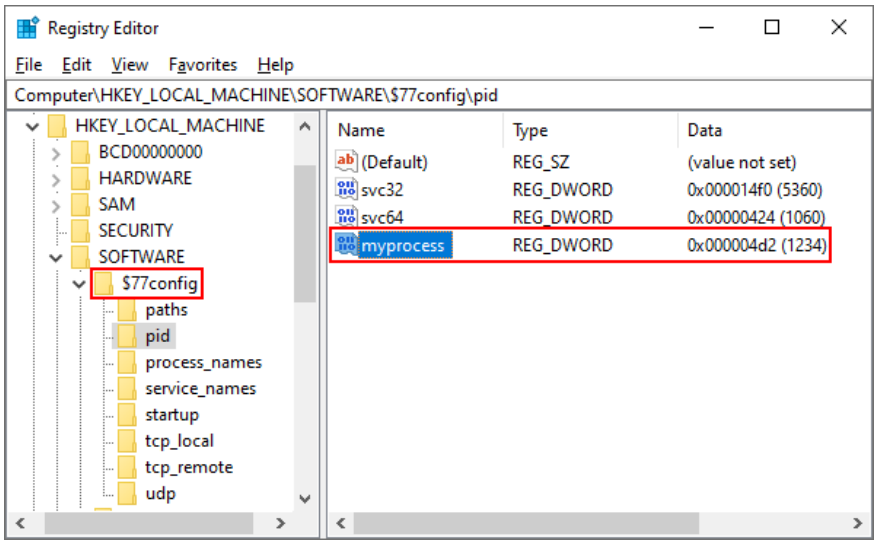
```
#define HIDE_PREFIX L"$77"
```

The hide prefix is a compile time variable. It specifies the beginning of a name that is hidden. This applies to the filenames of processes, to files and directories, and all other entities described in section 2.5. If the documentation mentions the string `$77`, it refers to the compile time variable that is used throughout the code. This is the single place to change the prefix. Additionally, this change must be applied in `GlobalAssemblyInfo.cs`.

Processes that are hidden by prefix cannot be injected with r77. The `DllMain` of r77 will return `FALSE`, if the process starts with the prefix.

2.7 Configuration System

The configuration system is stored in the registry under `HKEY_LOCAL_MACHINE\SOFTWARE\$77config`. The DACL of this registry key is set to allow full access to all users.



The configuration is read by r77 every 1000ms into a structure that holds following information:

- Array of hidden process ID's
- Array of hidden process names
- Array of hidden paths
- Array of hidden service names
- Array of hidden local TCP ports
- Array of hidden remote TCP ports
- Array of hidden UDP ports
- Array of startup paths

This data is used to hide entities based on custom configuration. Any process can write to the configuration system and does **not** require elevated privileges.

The name of the values is generally ignored. The values `$77config\pid\svc32` and `$77config\pid\svc64`, however, are reserved to the r77 service and should not be modified. They are created automatically when the r77 service starts.

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

2.7.1 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.

2.7.2 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.

2.7.3 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

2.7.4 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.

2.7.5 Local TCP Ports

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

2.7.6 Remote TCP Ports

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

2.7.7 UDP Ports

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

2.7.8 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. The files (typically executables) are started under the SYSTEM account.

2.8 Custom Startup Files

As described in section 2.7.8, it is possible to write paths of startup files (typically executables) 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 has to have the prefix, or it has to 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 4.1).

2.9 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 be used.
`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 are 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 `ControlPipeExample.cpp` demonstrates how to make r77 perform a ShellExecute. The Test Console can be used to test all existing control codes.

2.10 Enumeration vs. 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.

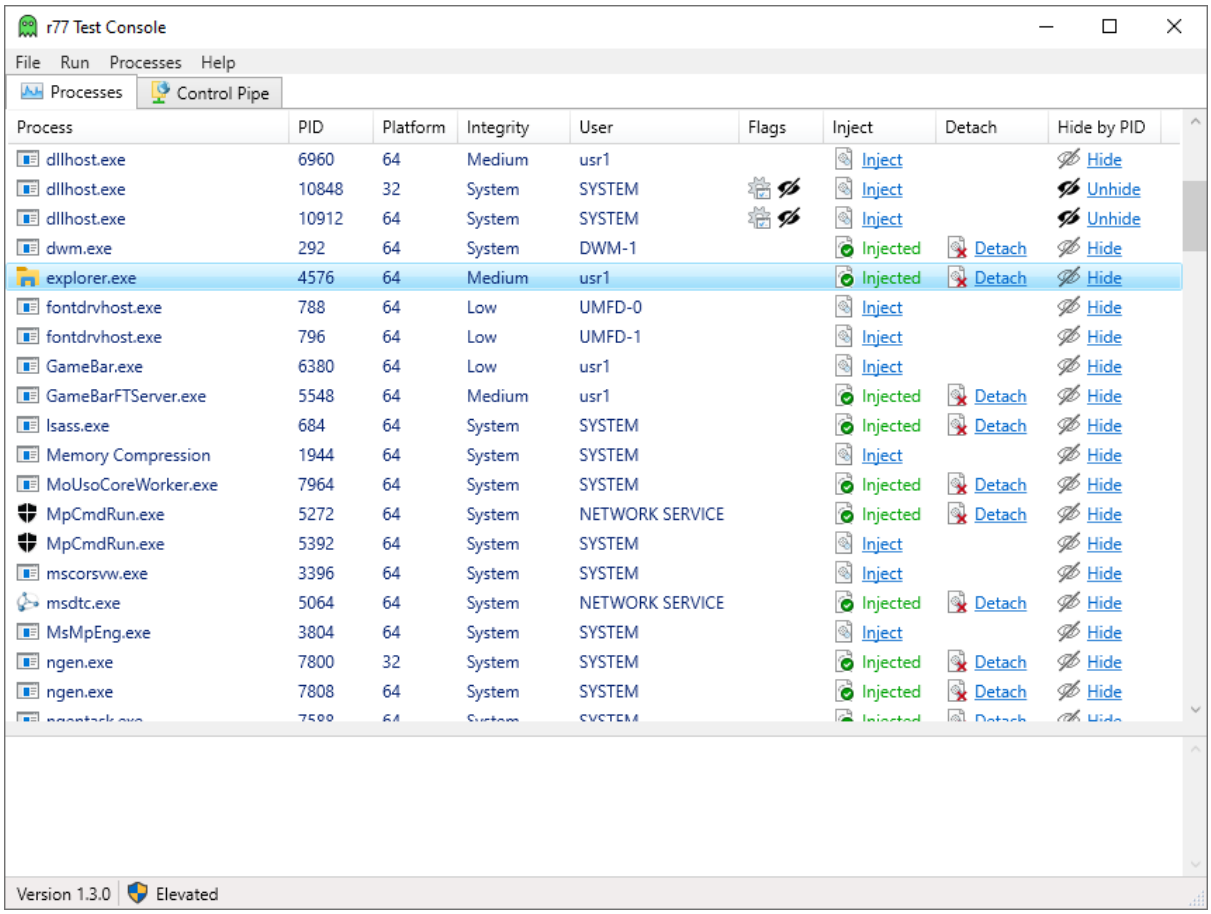
This is an issue that may be addressed in future releases of r77.

3 Test Environment

The standard, single file to deploy r77 is `Install.exe`. In addition, the test environment can be used to inject r77 to or detach r77 from individual processes. This is particularly relevant in the development process of r77. Also, it is a useful tool to test r77 during integration.

3.1 Test Console

`TestConsole.exe` can be used to inject r77 to or detach r77 from individual processes. The list of processes is not filtered by r77, because r77 does not inject the Test Console.



The process list shows, which processes are injected. The “Hide”-Link can be used to write a specific process ID into the configuration system.

The “Flags”-Column shows flags for a specific process.

Flag	Meaning
	The process is the r77 service process. It cannot be injected with r77.

	The process is an r77 helper process. It cannot be injected with r77. `TestConsole.exe` is a helper process, as well as `Helper32.exe` and `Helper64.exe`, which are invoked by the Test Console. Also, `Install.exe` and `Uninstall.exe` are helper processes. This is to avoid r77 being injected into them.
	The process ID is found in the configuration system. A task manager does not display this process. This includes the r77 service by default and can be extended to other processes. `Uninstall.exe` deletes the list of hidden process ID's.

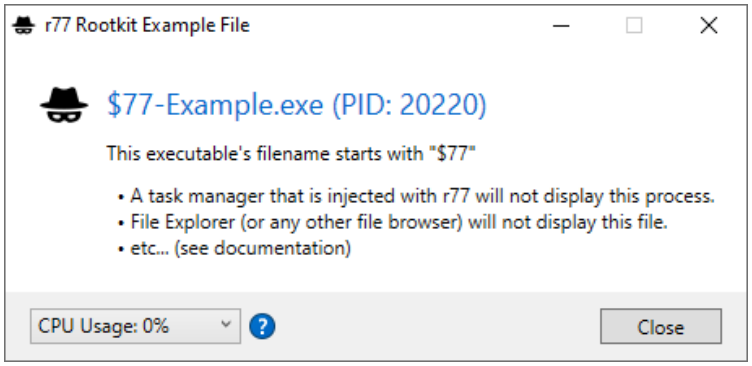
The Test Console is written in C#. This is a list of included files:

File	Purpose
BytecodeApi*.dll	Dependencies of `TestConsole.exe`.
Helper32.exe Helper64.exe	The Test Console uses these command line executables to: • Get a list of all processes. Because the process list in the Test Console contains r77 related information, the bitness of the executable that enumerates processes needs to match the bitness of the enumerated process. Each time the process list is refreshed, both executables are invoked and their output is parsed to populate the process list. • Inject a specific process or inject all processes. • Detach from a specific process or detach from all processes.
r77-x86.dll r77-x64.dll	The rootkit DLL's are included in the resources of `Install.exe` to allow full in-memory injection. However, `Helper32.exe` and `Helper64.exe` are loading these DLL files from the disk.

All above files are used by the Test Console exclusively and are not part of the r77 delivery.

3.2 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.

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 6 regarding issues with processor usage graphs.

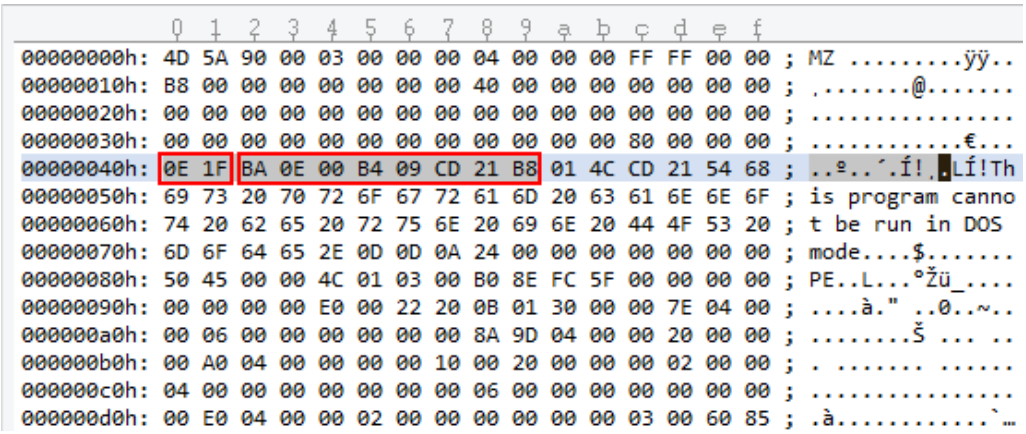
4 Implementation Details

This section describes implementation details, in addition to those mentioned in the above sections.

4.1 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 mapping and **not** the view of the file.

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. After `R77_SIGNATURE`, a function pointer to `Rootkit::Detach` is stored. 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 r77 service processes need to be identifiable to `Install.exe` and `Uninstall.exe`. This signature is written to the executable file at **compile time** to avoid injection of r77 into the service process prior to a signature being written. 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.

4.2 Compile Time Constants

Compile time constants are defined in both `r77api.h` and `GlobalAssemblyInfo.cs`. Changes must be applied in both files.

The r77 signatures are already mentioned in section 4.1. 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_NAME32`
= HIDE_PREFIX + "svc32"
`R77_SERVICE_NAME64`
= HIDE_PREFIX + "svc64"
	The name of the scheduled tasks that launch the r77 service processes. For scheduled tasks, a `.job` file is created, therefore the prefix is important.

`CHILD_PROCESS_PIPE_NAME32`
= "\\\\.\\pipe\\" + HIDE_PREFIX + "childproc32"
`CHILD_PROCESS_PIPE_NAME64`
= "\\\\.\\pipe\\" + HIDE_PREFIX + "childproc64"
	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 2.9.
`PROCESS_EXCLUSIONS` = { L"MSBuild.exe" }	Hardcoded list of processes that will not be injected.

4.3 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 and vice versa. As described before, a 32-bit executable cannot inject a 64-bit DLL into a 64-bit process. To solve this, 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. Either the 32-bit, or the 64-bit r77 service must be chosen to send the request to, based on the bitness of the created child process.

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.

4.4 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 4.4.7 about why this is a requirement.

The following chapters describe each function that is hooked.

4.4.1 NtQuerySystemInformation

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

4.4.2 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.

4.4.3 NtQueryDirectoryFile

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

4.4.4 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`.

4.4.5 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.

4.4.6 EnumServiceGroupW

This function is used to enumerate services.

4.4.7 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.

4.4.8 NtEnumerateValueKey

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

4.4.9 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.

4.5 AV Evasion Techniques

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

4.5.1 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.cpp`, the Powershell startup script is composed to contain code that performs this patch. This code is dynamically obfuscated when r77 is installed.

Note: The Powershell code must not contain any `Add-Type` cmdlets with C# code. It would invoke `csc.exe` (.NET compiler), which would drop a C# DLL to the disk.

Note: Bypassing AMSI is only required on Windows 10, as it is not supported in Windows 7.

4.5.2 DLL unhooking

Many EDR solutions implement hooks in `ntdll.dll`, and sometimes in `kernel32.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 using process hollowing. To evade detection of process hollowing, EDR hooks need to be removed.
- **r77 service:** Because the r77 service injects all running processes, EDR hooks must be removed here as well.

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.

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.

Note: If you are invoking `Install.exe` using process hollowing, the process that performs process hollowing should also unhook `ntdll.dll`. You can use the function `UnhookDll()` as an implementation reference. See: `r77api.h` and `Unhook.cs` for both the C# and C++ implementation.

5 Integration Best Practices

Including r77 into an existing project is simple and can be done in several ways.

5.1 Include Install.exe

Including the installer and executing it upon installation of your project is the preferred way. Executing the installer when r77 is already installed is supported. It will not update already injected processes; however new processes are injected with the new version of the rootkit DLL.

The installer can either be written to disk and executed, or it can be spawned using process hollowing. The process hollowing implementation needs to be written in the language of your project. If your project is written in C# or C++, the process hollowing implementation found in the r77 source code can be used as a reference. Otherwise, you have to write it yourself. It is required to perform 32-bit process hollowing, because the installer is a native 32-bit executable.

Executing the installer in-memory is an extra mile worth taking, because dropping the file on the disk will likely trigger AV detection.

If you encounter AV detection issues when using process hollowing to run `Install.exe`, you should perform DLL unhooking prior to performing process hollowing. It is your responsibility to evade AV/EDR detections to successfully run the installer. See section 4.5.2 on how to implement DLL unhooking when performing process hollowing.

5.2 Implement Installation Directly

An alternative option if you want more control over the execution flow is to implement the installer directly into the code of your project. For this, the behavior of `Install.cpp` must be replicated:

1. `InstallStager.exe` needs to be included in your project's resources.
2. `InstallStager.exe` must be written to both the 32-bit and the 64-bit registry key.
3. Both scheduled tasks need to be created and started.

Your implementation does not require a native executable. Make sure that your code handles Windows x86/x64 differences properly. For each r77 update, build the solution and take the `InstallStager.exe` file from the resources of the installer project. Make sure to check for changes in the code of `Install.cpp` and implement them in your project as needed. The source code of the installer itself is fairly short and commented.

Note: This is for advanced users only. Typically, including `Install.exe` as described in section 5.1 is sufficient.

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 Visual Studio binaries, such as MSBuild.exe are not working correctly when r77 is injected.
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 Windows apps 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.
Hiding of TCP and UDP rows does currently not work in the netstat command in `cmd.exe`. The exact mechanism used by netstat needs to be determined to hook the appropriate function.

7 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.
- Hide GPU usage
- Prohibit querying/opening/deleting/etc. operations on hidden entities (see section 2.10).
- More features for the control pipe:
    - Execute shellcode
    - Inject DLL into any process

8 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 repository or visit bytecode77.com/contact to get in touch.

Bug reports are only considered for supported operating systems (See section 1.1).

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 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 cannot be achieved is, if r77 would reside on the disk, which is not the case.
- Scan-time detection of `Install.exe`. This file should be executed using process hollowing as described in section 5. To run the installer in a fileless manner is your job.

9 Change Log

Version	Release	Changes
0.6.0	17.12.2017	**Beta release**
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
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.
1.1.0	11.04.2021	• Hide processes by name • Hide files, directories, junctions and named pipes by full path
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`.
1.2.1	20.06.2021	Several AV evasion techniques have been implemented (see section 4.5). • 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`.
1.2.2	31.08.2021	• Custom Startup Files (see section 2.8) to address the issue that Windows does not consider hidden files in startup, because Windows cannot “see” these files.

1.3.0		• 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 2.9).