SHELLTER PRO PLUS

Exclusive Features

```
 _____ __          _____            ____            ____  __
/ ___// /_  ___  / / / /___  _____/ __ \_________  / __ \/ /_  _______
\__ \/ __ \/ _ \/ / / __/ _ \/ ___/ /_/ / ___/ __ \/ /_/ / / / / / ___/
___/ / / / /  __/ / / /_/  __/ /  / ____/ /  / /_/ / ____/ / /_/ (__  )
/____/_/ /_/\___/_/_/\__/\___/_/  /_/   /_/   \____/_/   /_/\__,_/____/ 9.0

                    Codename: Eternal Nemesis              [x64]

                     www.ShellterProject.com

                History has a Nemesis for every sin.
                - Theodor Mommsen
```

Table of Contents

INTRO

This document provides an overview of the exclusive features added in Shellter Pro Plus.

Shellter Pro Plus introduces several new features that provide runtime evasion against AV and EDRs.

It is recommended to operate Shellter Pro Plus via our GUI application which provides access to all the features and offers a simple and intuitive way of operating our software for best results.

For more details about how to use these features, run Shellter with the -h argument, and/or take a look at the demos that are available through our official website.

! **Note:** If you run Shellter in Windows 11, you need to change the default terminal in Windows settings to "Windows Console Host". The new "Windows Terminal" that may be used by default has some compatibility issues with our console output. !

If you still need to know more about a specific feature, send an email to support@shellterproject.com.

LICENSE EXPIRATION CHECK UPDATES

Some extra updates regarding license expiration checks have been applied, that will now affect also the binaries where you inject your payloads.

! **Warning:** From version 8.5 onwards, license expiration checks will also be embedded inside the infected binary. !

Several advanced features and payload execution functions will check if the license used during infection has expired based on the date retrieved from the host where the binary is running.

When license expiration has been detected, several advanced features, such as dynamic unhooking of new loaded modules, will cease to work.

Additionally, any subsequent payloads that may be dormant, will not execute once the license validity date has lapsed.

Executing a binary for the first time on a host where the date indicates that the license used has expired will have no effect on target; meaning that no runtime evasion or other advanced features will be used and no payloads will ever run.

Please keep this in mind if you are about to start a security engagement that may be extended past the expiration date of your license.

SHELLTER PRO PLUS 9.X – ADDITIONAL FEATURES

MEMORY SCAN EVASION

Functions that are part of the relocatable code that handles all the advanced features, have been re-engineered in order to provide the ability to be dynamically decoded/re-encoded on the fly as required. While at rest, execute code permissions are removed from the corresponding memory allocations to further camouflage their purpose.

In addition to the relocatable code stubs protection, this feature also protects important data stored in data structures on runtime such as, but not limited to, function pointers. Everything protected by this feature will be fetched and decoded as needed, while the original copy inside the data structure will always remain encoded.

Function stubs protection is an optional feature and it can be enabled/disabled at will in order to give the ability the user to do some local testing using both cases, and to troubleshoot potential incompatibilities with advanced payloads.

On the other hand, any other data inside structures that is originally marked as protected, it will always be encoded even if the user disables the function stubs protection. Same applies also to all dynamically built strings which will be cleared off the current thread's stack memory after usage.

Note: This feature will remain silent if DBG/VM are enabled and either are detected on process startup. See ANTI-ANALYSIS CONCEALMENT I section for further information.

Overall, our ***'Memory Scan Evasion'*** feature provides an extra layer of defence against memory scanning practices that may be implemented by security solutions.

> **Note:** This feature only protects our own in-memory code and data.
> Our software is payload agnostic and intended to be used with payloads generated by various C2 frameworks. Once a payload has been decrypted in memory, it must implement its own self-hiding measures against signatures and byte pattern fingerprinting.

This feature is always enabled when operating our software in *'Auto mode'*.

In other operation modes including, *'Manual'* and *'GUI/CLI'*, the feature must be enabled by the user.

GUI option setting: Please go to 'Runtime Evasion' tab.

CLI argument: --MemScanEvasion

TOTAL RECALL

This feature was implemented in order to further eradicate in-memory traces of our code under certain circumstances.

There may be scenarios where none of the payloads will execute for various reasons such as, VM/DBG detection on process start, and/or failure to fetch remote AES keys for payload decryption.

When this happens, there is no reason to leave behind any traces of our code in memory.

This applies especially in ***'Stealth mode'*** where the infected EXE/DLL will continue to execute normally, and so the associated software may still run while no payload execution will take place.

When this is the case, the ***'Total Recall'*** feature will kick in and remove all of our executable code from memory, while cleaning up also memory areas that may store additional data.

> **Note:** This feature serves as an additional anti-analysis measure against sandboxes that may provide memory dumps of the examined process.

SHELLTER PRO PLUS 8.X – ADDITIONAL FEATURES

EXTENDED RUNTIME EVASION CAPABILITIES III

Several enhancements regarding runtime evasion have been implemented for this release.

We have also added the ability to remove suspicious access permissions from all proprietary memory allocations.

We are still working in this area to provide further flexibility and additional self-hiding abilities to the upcoming releases.

See: Memory Scan Evasion, Total Recall features added in version 9.

CONFIGURATION ENHANCEMENTS I

Some features configuration abilities have been upgraded by exposing more internal features through command line arguments, including GUI interface, and in Manual (interactive) operation mode. This allows to optionally enable/disable some runtime evasion features that were previously always enabled, when supported, by default. **Applies to version 8.2 and above.**

Currently, these include:

- ETW Evasion (--evadeETW)
- AMSI Evasion (--evadeAMSI)
- Native Imports Redirection (--redirectNativeImports)

There are obviously more internal features that apply for runtime evasion, which currently always apply when supported, that may also be exposed to the user for further configuration in the future.

Note: In Auto mode these features are enabled by default when supported. See ANTI-ANALYSIS CONCEALMENT I section for further information.

ANTI-ANALYSIS CONCEALMENT I

Some features will behave differently when a debugger (kernel-mode/user-mode) and/or hypervisor environment are detected on process startup. If either of those two features are enabled and triggered then some runtime evasion features will remain silent for concealment purposes. **Applies to version 8.2 and above.**

Currently, these include:

- ETW Evasion
- AMSI Evasion
- Native Imports Redirection
- Anti-DLL Load Monitoring
- Memory Scan Evasion (Added in v9.0)

Note: Native Imports Redirection only applies if at least one hook is detected on a native function (Nt*/Zw*) exported by Ntdll module. Otherwise, it will not be actioned even if it is enabled by the user.

> **Warning:** If you wish to test any of the features listed above in a VM environment, then make sure that hypervisor detection is disabled. In addition, if you have enabled kernel-mode debugger detection, then make sure kernel debugger is disabled in the OS.

ENHANCED APPLICATION DLL BACKDOORING I

This feature enables the user to perform application backdooring, while maintaining its original functionality, through one of its proprietary DLLs in a more efficient and reliable manner. When **'DLL Load Monitoring'** is activated, Shellter will monitor an application for DLL loading and exports calling events. No modifications are made to any binaries at this stage. This feature is available in *'Auto'* and '*CLI/GUI*' operation modes. Shellter must run either in a physical Windows host or inside a Windows VM. **Applies to version 8.3 and above.**

Command line arguments added: --monitorDllLoading/--MDLL

> **Note:** If a DLL is statically linked to the executable, then it will not be logged. If a DLL is logged, but no exported function is called, then it will not be displayed. See also "ENHANCED APPLICATION DLL BACKDOORING II" for further updates applied to this feature.

Once monitoring stage has finished, the following information is reported:

- **Full path of logged DLL.**
  - The full path is logged because an application may have multiple versions of a DLL that are loaded from different sub-directories.
- **Thread ID of the DLL loading event.**
  - This can be used to exclude DLLs that are being loaded by a new thread and only focus on those that were loaded by the thread id that was reported first; for reliability purposes.
- **First Export Called**
  - This is the first DLL-exported function that is called on runtime. When you infect the DLL, this is the export that you must set for DLL execution tracing.
- **Advanced features support.**
  - It reports that the DLL has advanced Shellter features support capabilities. This feature is meant to be used always with Stealth mode enabled. For this reason, any DLLs that do not support advanced features will not be shown to the user.
- **Signing status.**
  - This will report if the DLL contains a digital signature. The signature is not validated. It can be used to exclude signed DLLs and only target DLLs that were not originally signed.

Please note, this information only refers to embedded signatures. Integrity information of DLLs and other files may be also signed via catalog files. This method is commonly used for Windows system modules integrity checks.

Warning: Since the purpose of this feature is application backdooring through a DLL, it is necessary that you always enable Stealth mode when you inject your payload into the chosen DLL. Otherwise, the application will crash. Do not use any DLLs that do not provide advanced features support.

Although it is not recommended to use this feature with system DLLs, it is possible to do so in some cases by using a 'DLL-Hijacking-type' technique.

Warning: This may not work if you copy the infected system DLL across different Windows versions due to required dependencies that may be missing.

An application may dynamically load also system DLLs, directly, or through another DLL that was loaded previously. In that case depending on the DLL loading search path, the application may first try to load a system DLL from its own directory. This opens the possibility to have an infected copy of a system DLL inside the application's directory.

Normally, in order to find system DLLs that may be potentially hi-jacked you should use an application such as 'ProcessMonitor' and look for *'NAME NOT FOUND'* results associated with *'CreateFile'* operations.

Once you have found potential candidates, you can copy those into the appropriate directories of the application as shown in *'Process Monitor'* and repeat the DLL load monitoring with Shellter in order to find out if any of those can be used for infection.

In addition, when you choose the target DLL to infect, either an application-proprietary or a system one as described above, you must make sure that the infected copy may not be shared by multiple processes. This is simply because another process loading the infected DLL, may call at first a different export function causing that other process to crash or to execute again the same payload if it behaves in the exact same way as the one you had targeted originally.

Warning: The infected DLL must not be shared by multiple processes at any point in time. This is something that you will have to verify yourself.

Finally, you may come across some caveats when trying to trace the chosen DLL. Specifically, Shellter may report that the DLL was not loaded correctly because some dependencies are missing. This is not a Shellter bug, and there is a specific reason why this might happen.

Warning: You may come across some caveats when trying to infect a DLL. A specific issue related to missing dependencies and a workaround are described below.

Some applications may store some DLLs in separate directories where not all their dependencies are necessarily present. This is not an issue when the process runs because it can set the working directory accordingly so that all dependencies will be loaded correctly on runtime.

However, when Shellter attempts to trace the chosen DLL, it will try to load it from the location where it

is stored at rest. Since it has no knowledge of how an application may decide to load its modules on runtime, you may come across the aforementioned issue.

If this happens, there is a workaround that can be used to work with a specific DLL temporarily in order to perform the execution tracing and inject a payload. You can try to copy the target DLL to a directory where all its dependencies are located and repeat the infection attempt.

There is a demo on our website which demonstrates the required steps to use this great feature effectively.

Enhanced Windows built-in Modules Compatibility

Shellter Pro Plus will now recognise the special imports mechanism used by Windows built-in native modules, thus effectively allowing advanced features usage with those that support them. **Applies to version 8.4 and above.**

Several native Windows built-in modules, including executables and DLLs, have a special imports mechanism that goes through some proxy DLL forwarders using a naming convention such as *"api-ms-win-core-*"*.

What that means, is that a Windows built-in DLL, for example *"iertutil.dll"* may import a function such as *"MapViewOfFile"* through the *"api-ms-win-core-memory-l1-1-0.dll"* module which will actually return an IAT pointer to the module (i.e., *"kernel32/kernebase.dll")* that actually exports that function.

The Shellter's imports parser will normally look for such functions that are imported by the clean PE binary target that we wish to inject our payloads to, and will report if an appropriate combination of such APIs is imported. This allows Shellter Pro Plus to use advanced features with a binary, such as *"Stealth Mode"*, *"DBG/VM Detection"*, *"AES-128 Encryption"*, and so on.

In earlier releases, the imports parser would not recognise such DLL function import forwarders and so it would always report that the target PE modules making use of them, have no support of advanced Shellter features, even if that was not the case.

There is a demo on our website which demonstrates the required steps to use this great feature effectively.

Enhanced Application DLL Backdooring II

Further enhancements have been applied to *the "ENHANCED APPLICATION DLL BACKDOORING I"* feature in order to facilitate the discovery of hi-jackable system modules by the target application. **Applies to version 8.5 and above.**

Shellter will now also log DLLs that are loaded from *"C:\Windows*"* sub-directories and will copy them temporarily inside the target application's folder while triggering automatically a second stage of the **'DLL Load Monitoring'** feature.

! **Warning:** In order to complete successfully the second stage of the **'DLL Load Monitoring',** the target application's directory must be writable by the Shellter process. !

Once the second stage of the **'DLL Load Monitoring'** has been completed, Shellter will delete all system modules that were copied under the target application's directory.

Finally, it will report to the user not just which proprietary DLLs were dynamically loaded by the target application, but also any system module that was loaded from inside the application's directory, which means that it can be infected by Shellter and used potentially via the usual hi-jacking method.

! **Warning:** When copying a system DLL module to the Shellter's directory in order to inject your payload to it, make sure that you temporarily rename it to avoid share access conflicts with our GUI application and/or Shellter itself. !

Targeted Runtime Evasion

We have introduced the ability to change the behaviour of some runtime evasion features based on detecting specific security solutions that may be installed on the target host. **Applies to version 8.5 and above.**

This change of behaviour is handled automatically by the code that is bundled with the payloads of your choice and aims to enhance further the evasion capabilities as well as the compatibility of the infected binaries with some special modes of various security solutions.

Since this is something that we will have to manage in a reactive manner, the following list will be updated accordingly.

Currently, the following product will trigger this feature:

- **CrowdStrike Falcon Security Sensor**
  - An incompatibility with the highest-security-level, aka "aggressive mode" was reported by one of our customers. This was caused by a change in the way this security solution reacts to certain events, and was not a detection in itself. Upon detecting this product, our code will now trigger a switch of behaviour in order to continue execution of the infected binary without any unwanted artefacts. This will happen at all security levels because features can be moved around by their developers.

Shellter Pro Plus 7.x – Additional Features

Extended Runtime Evasion Capabilities II

This release brings additional enhancements to the runtime evasion capabilities by adding code that monitors in real time for modules being loaded into the process.

Advanced payloads usually require to load additional modules in order to complete several tasks. Since security software will commonly monitor these events through kernel-mode callbacks, it may optionally hook additional modules beyond the usual suspects such as Kernel32 and Ntdll.

The code that is bundled with your chosen payloads, will now monitor for newly loaded modules that are by default found under the 'KnownDlls' directory and will make sure that these will also be checked for hooks and other artefacts.

SHELLTER PRO PLUS 6.X – ADDITIONAL FEATURES

EXTENDED RUNTIME EVASION CAPABILITIES

This release includes multiple updates towards runtime evasion against various techniques used by security software to intercept searching and calling system functions; especially those exported by Kernel32, KernelBase, and Ntdll DLLs.

This particular entry might not look too fancy, but the internal updates associated with this enhancement offer a lot more than meets the eye.

AMBUSH PAYLOAD EXECUTION

When this feature is enabled, it will set all injected payloads into hibernation until the specified benign DLL is loaded by the process.

This special feature offers the ability to perform deep infection of an application while evading automated analysis systems, and execution emulation.

It also offers the capability to simulate threat actors that compromise supply chains in order to distribute their malware though legitimate software that have previously infected by using a more advanced technique.

! **Warning:** In order to take advantage of this feature, the specified DLL must not be statically linked to the target PE binary and/or to any other modules that are statically linked to it. It must be a DLL that is dynamically loaded when the user interacts with a specific feature of the infected application. !

There is a demo on our website which demonstrates the required steps to use this great feature effectively.

ANTI-DLL LOAD MONITORING

This feature removes user-mode registered callbacks that may be set by modules injected by security software inside the process in order to monitor for new DLL loading events.

! **Warning:** If a debugger or VM environment are detected, then this feature will remain silent for concealment purposes. !

SHELLTER PRO PLUS 5.X - ADDITIONAL FEATURES

ADVANCED DEBUGGER DETECTION (KM + UM)

Shellter Pro Plus offers the ability to insert additional code that is able to detect both kernel and user mode Windows debuggers. This feature is enabled by default in 'Auto' mode for both types.

If you use Shellter Pro Plus via 'Manual' mode and/or through our GUI/CLI interfaces you can choose the level of detection in order to target one or the other type only if you wish.

If user mode debugger detection is enabled, the added code will check if the process is being debugged by a user mode debugger.

If kernel mode debugger is enabled, the added code will check if the kernel debugger is currently enabled in the OS, which is normally disabled by default.

! **Warning:** If a debugger is detected, then the payloads will not run. !

If remotely-fetched AES-128 key/iv pair feature is used then there will be no attempt to download this data.

This feature can be combined with 'Decoy Payloads' feature.

ADVANCED VM/SANDBOX DETECTION

Shellter Pro Plus offers the ability to insert additional code that is able to detect both type-1 and type-2 hypervisors.

In a few words, both bare metal such as ESXI, Hyper-V, KVM etc…, and the usual suspects such as VMWare Workstation, Oracle VirtualBox and so on.

If you use Shellter Pro Plus via 'Manual' or 'Auto' modes then you can only choose a VM detection profile based on certain hardware resources available to the system, such as number of CPU cores, available RAM etc….

If you use Shellter Pro Plus via our GUI/CLI interfaces you can choose additional options, including our 'SecretSauce' which is able to detect both hypervisor types by using low level checks.

! **Warning:** If a hypervisor is detected, then the payloads will not run. !

If remotely-fetched AES-128 key/iv pair feature is used then there will be no attempt to download this data.

This feature can be combined with 'Decoy Payloads' feature.

DECOY PAYLOADS

Shellter Pro Plus offers the ability to insert a payload that will be used as a decoy in case the Debugger/VM detections trigger.

This can be used to tamper with both automated analysis sandbox systems and with manual analysis of your binary.

The decoy payload can be as complex as you wish such as a reflective DLL that performs multiple actions, a payload that crashes/shuts down/re-boots the system, or just a simple stager that connects to a fake IP/Port.

> ! **Warning:** In order to use this feature, Debugger and/or VM detection must be enabled. !

You will then have to choose a minimum of 2 payloads to inject into the clean binary. Keep in mind that in this case the first payload will be the decoy and the rest will be any 'real' payloads that you wish to execute normally.

So, make sure that you don't use one of the good payloads as the first one if you enable this feature.

If remotely-fetched AES-128 key/iv pair feature is used then there will be no attempt to download this data. The decoy payload will be encrypted using a separate AES-128 key/iv pair which will be embedded inside the binary.

> ! **Warning:** If you don't enable the decoy payload feature and a Debugger/VM are detected, then none of the injected payloads will run. !

AES-128 Payload Encryption

Shellter Pro Plus encrypts all payloads with AES-128 (CBC mode) algorithm by using a key/iv pair that is generated randomly every time.

Normally, this pair is embedded inside the binary that will run your payloads, but there's also the option to fetch the decryption key/iv pair from a remote location.

This allows you to control for how long someone could potentially analyse your special payloads in case they get a copy of your binary. Once you have removed the keys from your server/endpoint, nobody will be able to decrypt and analyse them.

Two methods are currently supported:

1. HTTP/HTTPS URL
2. UNC

Shellter Pro Plus will ask you to supply the full remote path to the file that contains the AES key information data.

For example, *"https://myserver.com/dir1/dir2/key.aes"* or *"\\myserver.com\\dir1\\dir2\\key.aes"*.

A key file with the specified name will be saved locally in the working directory using the file name that you specified in the path (i.e., "key.aes"). This file must be placed at the exact remote path as specified previously.

When using a URL to fetch the data the following rules apply:

1. Can use both HTTP and HTTPS protocols.
    a. Redirections from HTTP to HTTPs are allowed.
    b. Redirections from HTTPS to HTTP are not allowed.
2. SSL Certificates checks.
    a. Validity dates.

b. Hostname given in the request.

You can also use UNC type paths to fetch the key from another host on the same or another network.

> **Warning:** Whichever method you use, you need to make sure that the file is accessible without requiring some sort of authentication.

Advanced Self-Unhooking

Shellter Pro Plus offers the ability to insert additional code that is able to remove hooks placed in native and other Ntdll functions. These hooks are commonly used by security software to monitor for abnormal behaviour before a system call is executed.

This additional injected code will run before your payloads in order to remove such 'redirections' to the monitoring code that is already loaded inside the process by the security software during process initialisation.

Advanced Heuristic Unlinking of AV/EDR modules

Shellter Pro Plus offers the ability to insert additional code that is able to unlink decoy modules placed inside the "*Process Environment Block*" (PEB) "*Loader Data*" linked-list structures that provide information for all loaded modules in the process.

Some security software may use those decoy modules in order to detect manual parsing of the export table of system modules, most commonly those of Kernel32 and Ntdll, by renaming the original modules and setting the decoys earlier in the linked list.

Advanced native Imports Redirection for loaded modules

Shellter Pro Plus offers the ability to insert additional code that is able to redirect *"Import Address Table"* (IAT) pointers of all loaded modules that hold addresses of native Ntdll functions in order to evade those hooks even if they get replaced by the security software.

Note: See ANTI-ANALYSIS CONCEALMENT I section for further information.

Advanced Stealth Payload Thread Creation

Shellter Pro Plus offers the ability to insert additional code that is able to kick of the main thread of each of the injected payloads by using legitimate code pivots that reside inside the already loaded modules of the process.

This helps to evade detections that look for suspicious thread start activity that occurs when the start address resides outside of a loaded module.

Advanced ETW Evasion

Shellter Pro Plus offers the ability to insert additional code that is able to blind the *"Event Tracing for Windows"* (ETW) functionality for the current process.

This is normally used in order to log specific events that occur inside a process which are then evaluated by the security software.

Note: See ANTI-ANALYSIS CONCEALMENT I section for further information.

Advanced AMSI Evasion

Shellter Pro Plus offers the ability to insert additional code that is able to blind the *"Antimalware Scan Interface"* (AMSI) of Windows.

This is normally used by Windows Defender Windows Defender and other security software in order to scan memory buffers of the process upon request.

Note: See ANTI-ANALYSIS CONCEALMENT I section for further information.

Advanced Self-Process and Payload Threads Protection

Shellter Pro Plus offers the ability to insert additional code that is able to modify the security information of the process executing your payloads as well the security information of their main threads that are kicked off by our code.

This can make manual analysis through debugging to become harder since some usermode debugger's process access requests may be blocked.

Advanced Native SysCalls-Based Runtime Evasion

Shellter Pro Plus does all the above by using native syscalls in order to evade hooks and other monitoring code injected in the process that executes your payloads.

In addition, extra low-level techniques are implemented in order to tamper with the ability of security software to detect the usage of direct syscalls.

Author: Kyriakos Economou
Insainted Ltd - www.ShellterProject.com

Twitter: @kyREcon / @ShellterProject
Email: kyrecon@shellterproject.com / sales@shellterproject.com