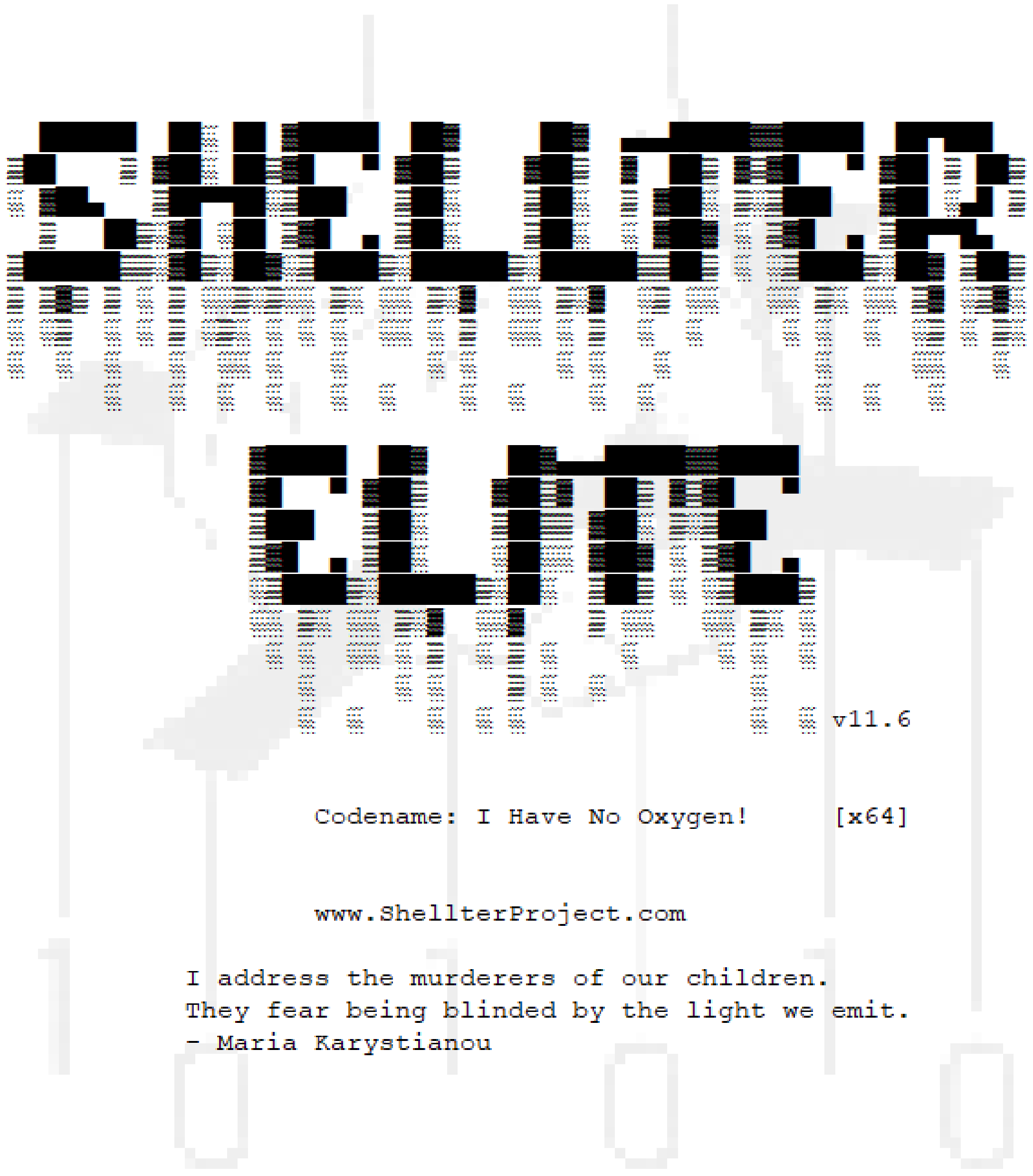
2026

SHELLTER ELITE

Exclusive Features



v11.6

Codename: I Have No Oxygen!   [x64]

www.ShellterProject.com

I address the murderers of our children.
They fear being blinded by the light we emit.
- Maria Karystianou

Table of Contents

INTRO

This document outlines the unique features introduced in Shellter Elite (formerly Shellter Pro Plus), in addition to all the functionalities available in the standard Pro edition of our software.[1]

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

It is recommended to operate Shellter Elite 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. !

! **Warning:** It is strongly recommended that you always check and verify the functionality of the output binaries in a controlled environment before you use them on a security engagement. !

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

[1] Shellter Pro Features

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 are also embedded inside the infected binary. Please read about the SELF-DISARM feature as well.

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.

Windows Version Support Updates

Note: **Beginning with Shellter Elite v11.3, support for Windows versions earlier than 10 is being deprecated in stages.** This change will likely impact WOW64 processes, while 64-bit binaries are expected to function as intended. However, we will not allocate resources to validate compatibility on deprecated Windows versions. **For further details regarding this decision, please refer to this article.**

Shellter Elite11.x – Additional Features

Shellter Elite v11.3 - Feature Updates

- We've refined our loader's runtime evasion capabilities against targeted EDR solutions, driven by valuable customer feedback.

Shellter Elite v11.1 - Feature Updates

- An important update has been applied to the REMOTE PAYLOADS and REMOTE AES KEYS features that allows for automatic proxy authentication if configured in the target host, i.e. with Kerberos.
- We 've further enhanced our loader's ability to evade detection during runtime execution.

Extended Runtime Evasion Capabilities IV

This release introduces multiple improvements to runtime evasion, aimed at optimizing different stages and functionalities of our loader.

Elite AMSI Evasion

An additional method has been introduced to bypass AMSI, allowing the process executing our code and payloads to evade detection more effectively.

Unlike conventional methods, this approach can bypass AMSI without altering memory permissions on the AMSI module, avoiding the need to patch either executable code or virtual function pointers.

Our code automatically determines and applies the most suitable AMSI evasion method based on various factors. However, this behaviour can be overridden if the user disables the "Enhanced AMSI Evasion" feature in the "Runtime Evasion" dialog or manually selects the standard ADVANCED AMSI EVASION method via the CLI.

! **Note:** See also ENHANCED AMSI EVASION section for further information. !

! **Warning:** It is **not** recommended to combine our AMSI evasion methods with others that may be offered by a C2 framework or custom ones that may be copied from other sources. !

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

CLI Argument: ***--EvadeAMSI-Enhanced***

Remote Payloads

Custom payloads now have the option to be hosted remotely, providing an alternative to the traditional method of embedding them directly within the infected binary.

This feature not only lowers the entropy of the infected binary but also enhances the security of your payloads against future analysis. By allowing remote payloads to be rendered inaccessible at any time, it ensures that previous payloads cannot be retrieved or examined at a later stage.

! Note: Remote payloads cannot be used as decoys. See DECOY PAYLOADS section for more information on that feature. !

Two methods are currently supported:

1. HTTP/HTTPS URL
2. UNC

CLI Arguments:

- ***--RemotePayloadPathURL <URL>***

- ***--RemotePayloadPathUNC <UNC>***

When the remote feature for a custom payload is enabled, Shellter compresses and encrypts the selected payload, saving it as a file with the same name but with the ".enc" filename extension.

In summary, if the custom payload is loaded from a file named "payload.bin," the encrypted file will be generated with the name "payload.bin.enc."

Next, you must provide the full remote path to the file containing the encrypted payload. This step ensures that the payload can be accessed and utilized as needed.

For example, *"https://myserver.com/dir1/dir2/test.bin"* or *"\\myserver.com\dir1\dir2\test.bin"*.

! Note: This is the filename of the encrypted payload that is saved locally. You will have to rename it to whatever name you gave to the remote path when copied to the remote host. For example, if you used one of the paths shown above, then the encrypted payload file must be renamed to *"test.bin"* when you copy it to that exact path. !

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.

Note: When using the URL method, you must define also a custom user agent by using the argument ***--RemotePayloadHttpUserAgent <UserAgent>*****.** Optionally, you can specify a custom HTTP(s) proxy by using the argument ***--RemotePayloadProxyUrl <PROXY_IP/DOMAIN:PORT>***. **The custom proxy must be accessible without requiring additional authentication. From version 11.1 automatic proxy authentication that may be configured in the target host, i.e. with Kerberos, is supported.**

Note: At present, if at least one payload is configured to be retrieved via a URL path, it is mandatory to enable the "remote AES keys" by URL feature, as outlined in the AES-128 Payload Encryption section. **However, this requirement does not apply when all remote payloads are accessed through UNC paths.**

Note: When payloads are retrieved via a URL path, the specified user agent and proxy settings will automatically apply to all payloads fetched using this method. These can be different from those used to fetch REMOTE AES KEYS.

In conclusion, remote payload path types and remote versus embedded payloads can be combined seamlessly. For instance, you can retrieve one payload through a URL path, another via a UNC path, and simultaneously include an embedded payload within the infected binary. This flexibility allows for diverse and adaptable configurations.

Warning: When you use remote payloads/decryption keys it is easy to forget sometimes to replace the files in the remote paths with the new files. Always make sure to update those files because each infected binary will only work with one set of encrypted payloads and/or decryption keys.

Payload Compression

By default, all payloads are now compressed before encryption. This optimization reduces the overall code size injected into infected binaries when embedded payloads are utilized, enhancing efficiency.

Additionally, remote payloads will also undergo compression before encryption. The compressed and encrypted payloads will then be saved to a file, which can later be retrieved from another host.

GUI option setting: Please go to 'Payloads' tab.

CLI Argument: --Disable-Payload-Compression

Increased Allowed Payload Size

The maximum allowable size for custom payloads has been increased from 4MB to 25MB. This enhancement accommodates the use of larger payloads, such as those generated by certain C2 frameworks, offering greater flexibility for users.

Note: For payloads exceeding a few hundred KB, it's advisable to leverage the REMOTE PAYLOADS feature. This enables runtime retrieval rather than embedding the payload directly into the infected binary, effectively preserving lower entropy and minimizing its detectability.

Force Preload System Modules II

The default 'basic' and 'networking' modules list of FORCE PRELOAD SYSTEM MODULES feature, have been revised.

Additional modules have been included to replace certain modules that would have otherwise been automatically loaded as dependencies of the ones already listed.

WARNING: When using this feature alongside AMBUSH, make sure the updated list excludes the module name you configure for that feature.

CallStack Scan Evasion

Modifies call stack information to evade detection mechanisms that rely on Kernel ETW (Event Tracing for Windows) data, ensuring improved stealth against such monitoring rules.

This feature works in conjunction with MEMORY SCAN EVASION to provide a more secure approach for executing syscalls—both directly and indirectly—calling various functions, and performing tasks such as loading additional system modules from disk.

Our newly introduced FORCE PRELOAD SYSTEM MODULES feature leverages this capability.

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

! **Note:** Enabling this feature will automatically activate MEMORY SCAN EVASION, ensuring protection for our in-memory code execution. As our software is payload agnostic, it is designed to work with payloads generated by a variety of C2 frameworks. After a payload is decrypted and executed, it can independently apply its own techniques to counter call stack analysis. This separation provides flexibility while maintaining robust evasion capabilities. !

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

CLI Argument: *--CallStackScanEvasion*

Force UnHooking System Modules via File-Mappings

This feature provides an alternative approach to ADVANCED SELF-UNHOOKING, effectively bypassing kernel mode callbacks inserted by EDRs to monitor new module loading events.

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

CLI Arguments:

- ***--ForceNtdllDecoyFileMapping***
  Use this method to only map a copy of NTDLL.DLL module temporarily.
- ***--ForceDecoyFileMapping***
  Use this method to temporarily map a copy for all loaded system modules.

  Starting with *version 10.1*, the second option will attempt to unhook all system modules being loaded, and not just those that are listed inside the *"KnownDlls"* section objects directory.

! **Note:** Loaded system modules unhooking only targets important system modules that are also listed inside the *"KnownDlls"* section objects directory, unless you specify the ***--ForceDecoyFileMapping*** *method*. **This feature will remain silent if the infected binary is executed in Windows versions earlier than Windows 8.** !

WARNING: Enabling this feature will automatically disable NATIVE IMPORTS REDIRECTION. This normally isn't an issue, because this feature is generally more desirable.

Force Preload System Modules

This feature offers a more secure approach for preloading system modules frequently utilized by C2 beacons and other third-party payloads. These modules support a range of system-level operations, such as establishing network connections, interacting with the Windows Registry, and performing cryptographic tasks, ensuring reliability and enhanced evasion capabilities.

To summarize, this feature ensures that all required modules are preloaded before the payload begins execution, guaranteeing they are readily available whenever the payload needs them.

Note: The load order of system modules that are included in our predefined modules list, will always be randomised. **Enabling this feature via CLI requires to enable CALLSTACK SCAN EVASION first.**

WARNING: If AMBUSH is enabled, ensure that the selected benign DLL module is not loaded through this feature, as doing so would undermine the purpose of using AMBUSH. Furthermore, if a conflicting DLL is loaded prior to AMBUSH being fully prepared, it may prevent the payload from executing altogether.

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

CLI Arguments:

- ***--Force-PreloadModules-Basic***
  Preloaded Modules: *MSWSOCK.DLL, WS2_32.DLL, WININET.DLL, WINHTTP.DLL, URLMON.DLL, WTSAPI32.DLL, GDI32.DLL, , ADVAPI32.DLL, NETAPI32.DLL, WINTRUST.DLL, CRYPT32.DLL, BCRYPT.DLL, NCRYPT.DLL, CRYPTSP.DLL, SECUR32.DLL, USERENV.DLL*

- ***--Force-PreloadModules-Networking***
  Preloaded Modules: *MSWSOCK.DLL, WS2_32.DLL, WININET.DLL, WINHTTP.DLL, URLMON.DLL*

- ***--Force-PreloadModules-Custom <modules_list>***
  Allows the user to define a custom list of system modules to preload before the payload execution begins. The maximum number of modules that can be set is 16 and the total length cannot exceed 256 characters.

Note: Only one of the options above can be enabled at the time.

Utilizing a custom system modules list is incompatible with specifying an EDR profile via the --Edr-Profile-* parameters. If you prefer to define your own list of system modules, you can manually include the relevant arguments provided under the profiles in TARGETED RUNTIME EVASION II section. This allows you to apply the same runtime evasion configurations against one of the specified EDR software solutions effectively.

Note: When setting a custom modules list, all module names must be separated with ';' even if it's a single module list or the last one in the list (i.e. wininet.dll;userenv.dll;). Modules must be present inside System32/SysWoW64 directories and the full name, including the ".dll" file name extension, must be set. **It is recommended to always include the networking-related system modules listed above.**

Enhanced AMSI Evasion

The ADVANCED AMSI EVASION feature has been refined with the inclusion of a new method that circumvents the need for in-memory patching of executable code within the AMSI module, as well as avoiding the use of hardware breakpoints. This enhancement ensures a more efficient and discreet evasion strategy.

This method will now be the default when using our software in interactive mode (Auto/Manual).

It is possible to choose between this and the original one via the GUI/CLI interfaces.

Note: See also ELITE AMSI EVASION section that was introduced in version 11, and ANTI-ANALYSIS CONCEALMENT I section for further information.

WARNING: Combining our AMSI evasion methods with those provided by a C2 framework or custom methods from external sources **is not** recommended. Doing so could lead to conflicts or unintended behaviours, potentially diminishing the effectiveness and reliability of the evasion techniques.

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

CLI Argument: *--EvadeAMSI-Enhanced*

Enhanced Memory Scan Evasion Polymorphism

Our MEMORY SCAN EVASION capabilities have been significantly improved with the introduction of a dynamic key mechanism for on-the-fly encoding/decoding of protected code and data in memory.

Previously, every infected PE file was assigned a unique key for encoding and decoding protected code and data. However, this key remained static across every execution of the binary, limiting the variability in its protection mechanism.

With this update, a dynamic key mechanism is introduced. Once the memory scan evasion feature is activated, the key dynamically changes during each execution. As a result, the protected code and data in memory will appear different every time, even when the same binary is executed. This enhancement significantly improves resilience against memory-based detection methods.

Self-Disarm

This feature allows users to define a specific timeframe, measured in days, after which the infected binary will automatically deactivate. If the binary is executed beyond this preset duration, no payloads will be triggered, ensuring a built-in limitation for its activity.

This rule ensures that the self-disarm timeframe applies uniformly to all payloads when multiple are injected. If a subsequent payload is set to execute beyond the specified timeframe, it will not be triggered, maintaining strict adherence to the predetermined deactivation period. This guarantees consistent behaviour across all payloads.

! **WARNING:** If the binary is initially executed within a valid date range but the TIMEBOMB feature is configured to extend beyond the self-disarm timeframe, no payloads will be triggered when the TIMEBOMB "countdown" concludes. This ensures that the self-disarm timeframe takes precedence, maintaining strict control over payload activation. !

Both the infection date and the self-disarm date are based on the local time of the system. Similarly, when the binary is executed on the target host, the self-disarm check will also rely on the host's local time. This ensures consistent enforcement of the timeframe across different environments.

! **Note:** The Self-Disarm feature offers a reliable way to ensure that binaries become harmless after the specified timeframe, providing an added layer of security. This can also function as a stealth mechanism if the binaries are subjected to analysis at a later stage. !

By default, the self-disarm threshold is set to 365 days, and users can configure this within a range of 1 to 365 days through the GUI or CLI, allowing for flexibility and control over the feature's behaviour.

GUI option setting: Please go to 'Miscellaneous'' tab.

CLI Argument: *--SelfDisarm <Num_Of_Days>*

Targeted Runtime Evasion II

The original TARGETED RUNTIME EVASION feature has been further enhanced to incorporate additional runtime evasion settings introduced after its initial release. These new settings expand its capabilities, offering more advanced techniques to counter various detection mechanisms effectively.

Additionally, users can now explicitly enable this feature based on the expected EDR present on the target host, provided this information is available. This enhancement grants greater flexibility and precision, allowing the feature to be tailored for optimal evasion in specific environments.

The following two EDR profiles are now available:

- **CrowdStrike Falcon** *(--Edr-Profile-CrowdStrike)*
  Will enable the following features: --MemScanEvasion, --ForceNtdllDecoyFileMapping

- **Elastic EDR** *(--Edr-Profile-Elastic)*
  Will enable the following features: --MemScanEvasion, --CallStackScanEvasion, *--ForceDecoyFileMapping*, --Force-PreloadModules-Networking.

TimeBomb

This feature enables users to define a delay, measured in milliseconds, before the execution of payloads begins. This provides additional control over the timing of payload execution, allowing users to fine-tune operations to suit specific scenarios or requirements.

The acceptable range for this feature allows users to specify a delay between 1 millisecond and 604,800,000 milliseconds (equivalent to 7 days).

Note: See also SELF-DISARM feature in order to combine these two appropriately.

GUI option setting: Please go to 'Miscellaneous' tab.

CLI Argument: --*TimeBomb <milliseconds>*

Slow Loris

This feature facilitates lazy loading by strategically introducing delays within frequently executed paths of our code.

This behaviour could assist in depleting default timing threshold values utilized by EDR software. These thresholds are often employed to track specific combinations of events that are logged through Kernel ETW monitoring or kernel mode callbacks. By doing so, it enhances evasion techniques, making detection by such monitoring mechanisms more challenging.

The acceptable range for this feature allows users to specify delays from 1 millisecond to 5,000 milliseconds (5 seconds). This range provides enough flexibility to fine-tune the timing for optimized lazy loading and evasion behavior.

WARNING: The specified delay operates in an additive manner, meaning that it accumulates each time the relevant code paths are executed. As a result, even the smallest delay value of 1 millisecond can lead to a cumulative delay of several seconds before the payloads are fully loaded and executed. This delay will persist and apply throughout the entire lifetime of the running process whenever those code paths are invoked

GUI option setting: Please go to 'Miscellaneous' tab.

CLI Argument: --*SlowLoris <milliseconds>*

Host Whitelisting

This feature ensures that payload execution is restricted to a designated physical host.

By tying execution to a specific hardware environment, it adds an additional layer of control and security, preventing payloads from running on unauthorized systems.

WARNING: If the binary is executed within a VM and the ADVANCED VM/SANDBOX DETECTION feature is either turned off or unable to detect the virtualized environment, the payloads will still proceed with execution as long as the C:\ volume serial number matches the specified value. This ensures that payload execution is tied to the designated environment, even in the absence of successful VM detection.

! **Note:** To retrieve the C:\ volume serial number of the host for your tests, open a command prompt and execute the command "vol". Then, input the retrieved serial number (excluding the hyphen) into the required argument. Note that serial numbers composed entirely of zeros (e.g., 00000000) are not permitted for this purpose. !

GUI option setting: Please go to 'Miscellaneous' tab.

CLI Argument: *--host-*

white-list-C-volume-serial <serial>

VM WHITELISTING

This feature provides the ability to whitelist a specific virtualized host, enabling users to test infected binaries without needing to completely disable VM detection mechanisms. **This functionality is available starting from version 10.1, offering greater flexibility and control during testing scenarios.**

This feature is seamlessly integrated with ADVANCED VM/SANDBOX DETECTION, ensuring that payloads do not execute on unauthorized virtualized hosts. At the same time, it allows you to perform thorough testing within a designated VM of your choice, combining robust security with flexibility for controlled testing scenarios.

This feature plays a crucial role in safeguarding your binaries during testing. By allowing controlled testing in whitelisted virtualized hosts, it minimizes the risk of binaries being exposed or compromised, especially in scenarios where security software uploads them to the cloud for automated sandbox analysis. This ensures both security and flexibility for comprehensive testing.

In summary, by integrating this feature with ADVANCED VM/SANDBOX DETECTION, you can comprehensively test the infected binary within a specified VM of your choice. This setup essentially replicates a scenario where VM detection is inactive or disabled, allowing for controlled testing while maintaining security protocols against unauthorized virtualized hosts.

! **Note:** To retrieve the C:\ volume serial number of the host for your tests, open a command prompt and execute the command "vol". Then, input the retrieved serial number (excluding the hyphen) into the required argument. Note that serial numbers composed entirely of zeros (e.g., 00000000) are not permitted for this purpose. !

GUI option setting: Please go to 'VM Detection' tab.

CLI Argument: *--vm-white-list-C-volume-serial <serial>*

REMOTE AES KEYS FETCHING WITH CUSTOM PROXY

This feature permits the definition of a custom proxy, which will be used to retrieve remotely stored AES keys essential for payload decryption and execution. By utilizing a proxy, it not only conceals the web server hosting these keys, thereby bolstering OPSEC capabilities but also ensures compatibility with corporate environments where custom proxy settings are employed. **This functionality is available starting from version 10.1, offering enhanced security and adaptability.**

For more details, about payload encryption please review the AES-128 PAYLOAD ENCRYPTION section.

! **Note:** Proxy URLs must use the HTTP(S) protocol and optionally include also a port number when necessary (i.e. http(s)://DOMAIN/IP:PORT). **Ensure that the custom proxy is accessible without requiring additional authentication. From version 11.1 automatic proxy authentication that may be configured in the target host i.e. with Kerberos, is supported.** !

GUI option setting: Please go to 'Payload Encryption' tab.

CLI Argument: *--RemoteAesKeyProxyUrl <URL>*

Shellter Pro Plus 9.x – Additional Features

Memory Scan Evasion

Functions within the relocatable code responsible for advanced features have undergone re-engineering to enable dynamic decoding and re-encoding as needed during runtime. When these functions are inactive, execute code permissions are deliberately removed from their respective memory allocations. This additional layer of security camouflages their purpose, making them more resistant to detection or analysis.

Beyond safeguarding relocatable code stubs, this feature extends its protection to critical runtime data stored in structures, such as function pointers. All protected elements are dynamically fetched and decoded only when needed. The original data within the structure stays encoded at all times, adding an extra layer of security and making unauthorized access or analysis significantly more challenging.

Function stubs protection is designed as an optional feature, allowing users to enable or disable it as needed. This flexibility empowers users to perform local testing under both scenarios—protected and unprotected—facilitating thorough evaluation and troubleshooting of potential incompatibilities with advanced payloads.

Any data within structures that is initially designated as protected will remain encoded at all times, regardless of whether the user disables the function stubs protection. Similarly, dynamically built strings are handled with additional care—they are cleared from the current thread's stack memory immediately after use. This approach ensures that sensitive information is consistently safeguarded and minimizes the risk of exposure during runtime.

> **Note:** This feature is designed to remain inactive if DBG/VM are enabled and either are detected on process startup. See ANTI-ANALYSIS CONCEALMENT I for further information.

In summary, our 'Memory Scan Evasion' feature bolsters security by offering an additional layer of defence against memory scanning techniques commonly employed by security solutions. By implementing advanced protection mechanisms, this feature significantly reduces the risk of detection or analysis, ensuring more resilient operations.

> **Note:** This feature is specifically designed to safeguard only the in-memory code and data of our own software. As the software is payload agnostic, it is compatible with payloads generated by various C2 frameworks. However, after a payload is decrypted in memory, it becomes the responsibility of the payload itself to employ self-hiding mechanisms.
> 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 designed to strengthen security by further eliminating in-memory traces of our code under specific conditions. By doing so, it minimizes the potential footprint left in memory, making detection or analysis more challenging for adversaries or security tools.

In certain scenarios, payloads may fail to execute due to specific conditions. These could include VM/DBG detection on process start, and/or failure to fetch remote AES keys for payload decryption.

In such scenarios, it becomes essential to ensure that no traces of our code remain in memory. This approach reinforces security by reducing the likelihood of detection or analysis, maintaining the integrity of the software even in non-operational circumstances.

This is particularly significant in ***'Stealth mode'*** where the infected EXE/DLL operates as usual, allowing the associated software to function without any payload execution occurring. By maintaining normal execution while leaving no payload-related traces, this mode further minimizes detection risks and reinforces operational secrecy in non-execution scenarios

In such scenarios, the ***'Total Recall'*** feature is activated to systematically remove all executable code associated with the software from memory. Additionally, it ensures that any memory regions storing supplementary data are thoroughly cleaned up. This meticulous approach enhances stealth and minimizes the possibility of leaving behind traces, even in cases where no payload execution occurs.

! **Note:** This feature acts as an extra layer of anti-analysis protection specifically designed to counter sandboxes that might generate memory dumps of the examined process. !

Enhanced Application DLL Backdooring III

The feature for monitoring a target binary's DLL load events (ENHANCED APPLICATION DLL BACKDOORING I) has been further refined, improving its ability to successfully backdoor the target via one of the dynamically loaded modules during runtime.

This enhancement allows users to specify command line arguments that the target binary would typically use when launched after installation. For instance, if a legitimate binary is optionally installed as a service, it might require distinct arguments to operate correctly.

In essence, this update enables monitoring for DLL load events by utilizing the same command line arguments that the target binary is expected to use during its normal execution. This ensures consistency in runtime behaviour, aligning with how the process or service would typically operate on the target system. **This functionality has been available since version 9.1.**

Infected Binary Watermarking I

This update introduces the ability to optionally append a custom string watermark to the infected binary. This watermark can serve as a unique identifier for the client, useful both during and after a security engagement. **This functionality, enhancing traceability and accountability, has been available since version 9.1.**

CLI Arguments:

- ***--Watermark-Gen-Random***
  Appends a randomly-generated watermark string to the infected PE file.

- ***--Watermark-Set-Custom <watermark_string>***
  Appends a custom watermark string to the infected PE file.

Moderate Polymorphic Junk Code Execution Time

The capability to include this type of code has been an integral part of the software from the beginning. It serves as an effective countermeasure against emulation engines.

This technique does not rely on traditional 'Sleep' methods. Instead, it employs randomly generated code sequences designed to overwhelm and exhaust an emulation engine. By doing so, the engine struggles to reach a state where it might observe events or behaviour it considers 'interesting,' effectively delaying or thwarting analysis efforts. This approach offers a more robust method of countering emulation-based analysis.

By default, this type of code executes more slowly when ‘Stealth’ mode is disabled. In such cases, there is no necessity to mimic the typical execution behaviour of the infected binary, allowing for more deliberate execution without compromising operational requirements. This distinction optimizes the approach based on the chosen mode, balancing functionality and security.

This update introduces an option to reduce the execution time of the randomized code to match the levels observed when ‘Stealth’ mode is enabled, even if that mode is not active. This adjustment provides greater flexibility and efficiency. **It has been available starting from version 9.1.**

Code Signing

This feature introduces the ability to automate the signing of infected binaries using customized self-signed certificates. **This functionality has been available since version 9.2.**

Note: This feature provides an alternative approach to watermarking by utilizing customized self-signed certificates, rather than the original method (INFECTED BINARY WATERMARKING I) which simply appends a string to the infected binary. All generated certificates are saved in current user's certificate store *(Cert:\CurrentUser\My\)*. This feature supports the automated deletion of a certificate once the signing process is completed.

WARNING: This feature is not compatible with the standard watermarking method and the original *'CertPlay'* features used to restore the digital signature of the binary if it was signed and/or embed one extracted from another binary. The GUI application will enable those accordingly.

In Auto/Manual modes, when this feature is enabled, the various properties of the generated self-signed certificate are entirely randomized. This ensures a unique and dynamic output every time, adding an extra layer of unpredictability and enhancing security measures.

Self-Signed Certificate Randomised properties:

- Key algorithm and length (RSA-2048/4096, ECDSA-256/384).
- Hashing algorithm (SHA-256/384/512) used to sign the certificate.
- Serial number.
- Issuer/Subject name.
- Domain name.

The digest algorithm (SHA-256/384/512) selection used for hashing the data during the signing process is also fully randomised.

When operating Shellter from CLI/GUI, the user can customise the **Issuer/Subject** and **Domain names**, as well as the **timestamp server URL** that will be used during the signing process.

CLI arguments:

- ***--Sign-With-Self-Signed-Cert***
  Enables automatic generation of a self-signed certificate and signing of the infected binary.

- ***--SubjectName-Self-Signed-Cert <subject_name>***
  Optionally set a custom Issuer/Subject name for the self-signed certificate.

- ***--DnsName-Self-Signed-Cert <dns_name>***
  Optionally set a custom domain name for the self-signed certificate.

- ***--FriendlyName-Self-Signed-Cert <friendly_name>***
  Optionally set a custom friendly name for the self-signed certificate. This is only useful when the self-signed certificate is saved in the current user's certificate store. If the user does not specify a friendly name, Shellter will

automatically generate one in the format: *SHELLTER_SELF_SIGNED_CERT_DATE_TIMESTAMP.*

- ***--Delete-Self-Signed-Cert***
  Optionally enable automatic deletion, from current user's certificate store, of the generated self-signed certificate at the end of the signing process.

- ***--Timestamp-Server-Url <url>***
  Optionally set a custom timestamp server URL to be used during the signing process. If this is not set, Shellter will use one from a predefined list. **Timestamping requires internet access.** If timestamping fails, it won't affect the signing process of the binary because this happens afterwards.

SHELLTER PRO PLUS 8.X – ADDITIONAL FEATURES

EXTENDED RUNTIME EVASION CAPABILITIES III

This release introduces multiple enhancements to runtime evasion techniques, further strengthening defences against detection and analysis. These improvements refine operational stealth and ensure even greater resilience when faced with runtime monitoring or inspection.

This update introduces the capability to remove any suspicious access permissions from all proprietary memory allocations.

We continue to focus on this area, aiming to introduce greater flexibility and enhanced self-hiding capabilities in future releases.

See: Memory Scan Evasion, Total Recall features added in version 9.
See: CallStack Scan Evasion feature added in version 10.

CONFIGURATION ENHANCEMENTS I

Certain features' configuration capabilities have been enhanced by exposing additional internal options via command line arguments. These upgrades also extend to the GUI interface and Manual (interactive) operation mode. Users can now optionally enable or disable specific runtime evasion features, which were previously always enabled by default, when supported. **These improvements have been available since version 8.2.**

Currently, these include:

- **ETW Evasion** *(--EvadeETW)*
- **AMSI Evasion** *(--EvadeAMSI)*
- **Native Imports Redirection** *(--RedirectNativeImports)*

Additional internal features related to runtime evasion are currently always applied when supported. However, there is potential for these features to be made configurable in the future, giving users greater control and customization options.

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

Anti-Analysis Concealment I

Certain features exhibit altered behaviour when a debugger (kernel-mode/user-mode) or a hypervisor environment is detected during process startup. If either of these detection features is enabled and triggered, specific runtime evasion functionalities will intentionally remain inactive to enhance concealment. **This behaviour adjustment has been effective since version 8.2.**

Currently, these include:

- ETW Evasion
- AMSI Evasion
- Native Imports Redirection
- Anti-DLL Load Monitoring
- Memory Scan Evasion (Added in v9.0)
- CallStack Scan Evasion (Added in v10.0)

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 plan to test any of the aforementioned features within a VM environment, ensure that hypervisor detection is turned off. Additionally, if kernel-mode debugger detection is active, it is essential to disable the kernel debugger in the operating system for proper testing. !

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 enabled, Shellter actively observes an application for DLL loading and exports calling events without making any changes to the binaries during this phase. This feature is accessible in both 'Auto' and 'CLI/GUI' operation modes. For proper functionality, Shellter must operate on a physical Windows host or within a Windows virtual machine. **This capability has been available**

starting from version 8.3.

CLI argument: *--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 to account for instances where an application may have multiple versions of the same DLL, each being loaded from different sub-directories.
- **Thread ID of the DLL loading event.**
  This functionality allows the exclusion of DLLs loaded by new threads, concentrating only on those DLLs that were loaded by the initial reported thread ID. This enhances reliability by prioritizing and analysing the activity tied to the primary thread.
- **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 must be 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.

After identifying potential candidates, transfer them to the appropriate application directories, as indicated by 'Process Monitor.' Then, utilize Shellter to repeat the DLL load monitoring process and determine if any of the candidates are suitable for infection.

Additionally, when selecting the target DLL to infect—whether it is application-specific or a system DLL as mentioned earlier—ensure that the infected copy is not shared among multiple processes.

This is because if another process loads the infected DLL, it may initially call a different export function. This could result in that process crashing or, if it behaves identically to the originally targeted process, re-executing the same payload.

! **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 might encounter certain challenges when tracing the selected DLL. For instance, Shellter may indicate that the DLL wasn't loaded properly due to missing dependencies. This issue is not a bug within Shellter but occurs for a specific underlying reason.

! **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. !

Certain applications might store DLLs in separate directories where not all dependencies are included. This usually isn't a problem during runtime, as the process can adjust the working directory to ensure that all necessary dependencies are correctly loaded.

When Shellter attempts to trace the selected DLL, it tries to load it directly from its stored location. Unlike the application's runtime behaviour, Shellter lacks the context of how the application manages its module loading. As a result, this limitation can lead to the previously mentioned issue.

If this occurs, a temporary workaround is available to facilitate execution tracing and payload injection. You can copy the target DLL to a directory that contains all of its dependencies, then attempt the infection process again.

There is a demo on our website that illustrates the necessary steps for utilizing this feature efficiently.

Enhanced Windows built-in Modules Compatibility

Shellter Pro Plus, **starting from version 8.4**, now supports recognizing the special imports mechanism used by Windows' built-in native modules. This enhancement enables the effective utilization of advanced features for those modules that support them.

Several native Windows built-in modules, like executables and DLLs, utilize a specialized imports mechanism. This process involves proxy DLL forwarders, often following a naming convention such as “api-ms-win-core-*.”

In essence, this means that certain Windows built-in DLLs, like *"iertutil.dll"*, may not directly import functions such as *"MapViewOfFile"*. Instead, they rely on a specialized mechanism where the function is imported through intermediary modules like *"api-ms-win-core-memory-l1-1-0.dll"*. This module acts as a proxy, ultimately redirecting to the actual exporting module, such as *"kernel32.dll"* or *"kernelbase.dll",* and returning an Import Address Table (IAT) pointer for the 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 previous versions, the imports parser was unable to recognize DLL function import forwarders. Consequently, it would inaccurately report that target PE modules utilizing such forwarders did not support advanced Shellter features, even when they actually did.

There is a demo on our website that illustrates the necessary steps for utilizing this feature efficiently.

Enhanced Application DLL Backdooring II

Starting from **version 8.5**, the ENHANCED APPLICATION DLL BACKDOORING I feature has received additional improvements. These enhancements streamline the process of identifying hijackable system modules within the target application, making the feature even more effective.

Shellter now includes functionality to log DLLs loaded from *"C:\Windows\*"* sub-directories. These DLLs will be temporarily copied into the target application's folder, which automatically initiates a second stage of **the 'DLL Load Monitoring'** feature.

! **WARNING:** To successfully complete the second stage of the **'DLL Load Monitoring'** process, it's essential that the target application's directory is writable by the Shellter process. !

After completing the second stage of the **'DLL Load Monitoring',** Shellter will automatically remove all system modules that were temporarily copied to the target application's directory.

Finally, Shellter will provide a report detailing not only the proprietary DLLs dynamically loaded by the target application but also any system modules loaded from within the application's directory. This indicates that these system modules can potentially be infected and utilized through the usual hijacking 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

Starting from **version 8.5**, a new capability has been introduced to adjust the behaviour of certain runtime evasion features. This adjustment is based on detecting specific security solutions that might be installed on the target host, allowing for more adaptive and effective functionality.

This behavioural adjustment is seamlessly managed by the code embedded within your chosen payloads. Its purpose is to further improve evasion capabilities while ensuring greater compatibility of the infected binaries with specific operational modes of various security solutions.

As this is a reactive process, the corresponding list will be continuously updated to reflect any necessary changes or additions.

Currently, the following product will trigger this feature:

- **CrowdStrike Falcon Security Sensor**
  - A customer encountered an incompatibility issue with the highest security level, referred to as *"aggressive mode"*. This was not due to detection but rather a change in how the security solution responds to specific events. To address this, the code now automatically triggers a behaviour switch upon detecting the product, ensuring the infected binary continues execution without generating unwanted artifacts. This adaptation applies across all security levels, as developers may shift features between levels.

SHELLTER PRO PLUS 7.X – ADDITIONAL FEATURES

EXTENDED RUNTIME EVASION CAPABILITIES II

This release introduces further improvements to runtime evasion capabilities by incorporating real-time monitoring for modules being loaded into the process. This enhancement allows for more dynamic responses and strengthens the evasion functionality.

Advanced payloads often need to load extra modules to accomplish various tasks. Since security software typically monitors such activities using kernel-mode callbacks, it may extend its hooks to include additional modules, beyond the commonly observed ones like Kernel32 and Ntdll.

The embedded code within your selected payloads now includes real-time monitoring for newly loaded modules located by default in the ‘KnownDlls’ directory. This enhancement ensures that these modules are thoroughly inspected for hooks and other potential artifacts, further strengthening the evasion capabilities of the payload.

Shellter Pro Plus 6.x – Additional Features

Extended Runtime Evasion Capabilities

This release introduces numerous advancements aimed at enhancing runtime evasion techniques. These updates specifically address methods used by security software to intercept the searching and calling of system functions, with a focus on those exported by Kernel32, KernelBase, and Ntdll DLLs.

Ambush Payload Execution

With this feature enabled, all injected payloads are placed into hibernation until the process loads the specified benign DLL. This ensures controlled activation of the payloads, allowing them to operate only after the required condition is met.

This advanced feature enables the capability to deeply infect an application while effectively bypassing automated analysis systems and execution emulation. This adds an extra layer of stealth and adaptability to the overall process.

This feature provides the ability to simulate threat actors compromising supply chains, enabling the distribution of malware through legitimate software that has been previously infected using advanced techniques. This simulation adds realism and sophistication to testing scenarios, emulating tactics used in actual cyberattacks.

GUI option setting: Please go to 'Payloads tab.

CLI argument: --ambush <DllName.dll>

WARNING: To utilize this feature effectively, the specified DLL must not be statically linked to the target PE binary or to any other statically linked modules. Instead, it must be a dynamically loaded DLL, triggered when the user interacts with a particular feature of the infected application. **See also: FORCE PRELOAD SYSTEM MODULES feature that was added in version 10 for compatibility notes between these two features.**

There is a demo on our website that illustrates the necessary steps for utilizing this feature efficiently.

Anti-DLL Load Monitoring

This feature effectively eliminates user-mode registered callbacks that might be established by modules injected by security software within the process. These callbacks are typically used to monitor new DLL loading events, and their removal enhances the stealth and functionality of the infected application.

! **WARNING:** If a debugger or VM environment is detected, this feature is designed to stay dormant and avoid any activity. !

Shellter Pro Plus 5.x - Additional Features

Advanced Debugger Detection (KM + UM)

Shellter Pro Plus includes a feature that can insert additional code capable of detecting both kernel-mode and user-mode Windows debuggers. This functionality is automatically enabled by default in the 'Auto' mode for both types.

When using Shellter Pro Plus in ‘Manual’ mode or through the GUI/CLI interfaces, you have the flexibility to customize the level of detection. This allows you to specifically target either kernel-mode or user-mode Windows debuggers, depending on your preferences or needs.

When kernel-mode debugger detection is enabled, the added code will verify whether the kernel debugger is currently active in the operating system. Since the kernel debugger is typically disabled by default, this check helps identify any unusual activity or debugging attempts at the kernel level.

! **WARNING:** If a debugger is detected—whether user-mode or kernel-mode—the payloads will remain dormant and will not execute. !

When the remotely-fetched AES-128 key/iv pair feature is utilized, no attempts will be made to download this data.

This feature can be combined with DECOY PAYLOADS feature.

Advanced VM/Sandbox Detection

Shellter Pro Plus introduces the capability to embed additional code that can detect both type-1 and type-2 hypervisors.

This capability covers both bare metal hypervisors like ESXi, Hyper-V, and KVM, as well as the more common ones like VMware Workstation and Oracle VirtualBox.

When using Shellter Pro Plus in 'Manual' or 'Auto' modes, VM detection profiles can only be selected based on specific hardware resources accessible within the system. These include factors like the number of CPU cores, available RAM, and similar attributes.

When using Shellter Pro Plus via the GUI/CLI interfaces, you gain access to additional options, including the innovative 'SecretSauce' feature. This advanced capability employs low-level checks to detect both type-1 and type-2 hypervisors, offering a comprehensive and precise approach to hypervisor detection.

! **WARNING:** If a hypervisor is detected—whether it's type-1 or type-2—none of the payloads will execute. !

When the remotely-fetched AES-128 key/iv pair feature is utilized, no attempts will be made to download this data.

This feature can be combined with DECOY PAYLOADS feature.

Decoy Payloads

Shellter Pro Plus includes the capability to embed a decoy payload, which activates when Debugger or VM detections are triggered. This ensures that the primary payloads remain concealed.

This functionality can be employed to interfere with both automated analysis sandbox systems and manual analysis efforts targeting your binary. By leveraging decoy payloads and advanced detection mechanisms, it adds an extra layer of complexity and misdirection.

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.

CLI Argument: *--decoy*

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

When injecting payloads into a clean binary using this feature, you must select a minimum of two payloads. The first payload will act as the decoy, while the subsequent payloads will be the 'real' ones you intend to execute under normal circumstances.

When the remotely-fetched AES-128 key/iv pair feature is utilized, no attempts will be made to download this data.

The decoy payload is protected by encryption, utilizing a distinct AES-128 key/IV pair that is embedded directly within the binary.

! **WARNING:** If the decoy payload feature is not enabled and a Debugger or VM is detected, none of the injected payloads will execute. !

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.

By default, the key/IV pair is embedded within the binary that executes the payloads. However, you also have the flexibility to configure the option for fetching the decryption key/IV pair from a remote location.

By remotely storing the key/IV pair and removing it from your server or endpoint after a set period, you gain significant control over how long anyone can analyse your special payloads, even if they acquire a copy of your binary. Without access to the key/IV pair, decryption and analysis become impossible, ensuring your payloads remain secure and shielded.

Two methods are currently supported:

3. HTTP/HTTPS URL
4. UNC

CLI Arguments:

- ***--AesKeyPathURL <URL>***

- ***--AesKeyPathUNC <UNC>***

Shellter Pro Plus requires you to provide the complete remote path to the file containing the AES key information data when using the remote key/IV pair option.

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:

3. Can use both HTTP and HTTPS protocols.
   a. Redirections from HTTP to HTTPs are allowed.
   b. Redirections from HTTPS to HTTP are not allowed.
4. SSL Certificates checks.
   a. Validity dates.
   b. Hostname given in the request.

! **Note:** When using the URL method to fetch the keys from a web server, it's possible to define also a custom user agent by using the argument *--HttpUserAgent <UserAgent>* !

! **Note:** Starting with version 10.1, it's also possible to fetch those keys through a proxy. See REMOTE AES KEYS FETCHING WITH CUSTOM PROXY section. !

You can also use UNC type paths to fetch the key from another host, whether it's on the same network or a different one

! **WARNING:** Whichever method you use, you need to make sure that the file is accessible without requiring any type of authentication. !

Advanced Self-Unhooking

Shellter Pro Plus includes the capability to insert additional code that can remove hooks placed in native Ntdll functions and other loaded system modules.

These hooks are often employed by security software to monitor for any unusual or suspicious behaviour before a system call is executed.

The additional injected code is designed to execute prior to your payloads, effectively removing any 'redirections' set by security software. These redirections are typically placed during process initialization to monitor system calls and behaviour.

! **Note:** The default unhooking method focuses on targeting key system modules that are listed in the *"KnownDlls"* section of the objects directory. **In order to unhook all loaded system modules, see FORCE UNHOOKING SYSTEM MODULES VIA FILE MAPPINGS.** !

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.

Certain security software can employ decoy modules to disrupt manual parsing of the export table in system modules like Kernel32 and Ntdll. By renaming the original modules and positioning the decoys earlier in the linked list, they create a misleading structure to detect or thwart analysis attempts.

Advanced Native Imports Redirection

Shellter Pro Plus has the capability to insert code that redirects the Import Address Table (IAT) pointers of all loaded modules that reference native Ntdll functions. This ensures that even if the hooks placed by security software are replaced, the redirection effectively bypasses them, maintaining stealth and operational integrity.

! **Note:** This feature is **not** compatible with FORCE UNHOOKING SYSTEM MODULES VIA FILE-MAPPINGS feature that was introduced in v10. See also ANTI-ANALYSIS CONCEALMENT I section for further information. !

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

CLI Argument: *--RedirectNativeImports*

Advanced Stealth Payload Thread Creation

Shellter Pro Plus provides the capability to kick off the main thread for each injected payload by utilizing legitimate code pivots within already loaded modules of the process.

By utilizing legitimate code pivots within already loaded modules to initiate the main thread for each payload, Shellter Pro Plus helps evade detections focused on suspicious thread start activities. These detections typically flag scenarios where the thread start address is located outside of a loaded module.

Advanced ETW Evasion

Shellter Pro Plus can inject additional code to blind the *"Event Tracing for Windows"* (ETW) functionality for the current process.

Event Tracing for Windows (ETW) is typically employed to log detailed events occurring within a process, enabling security software to evaluate and monitor specific behaviours.

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

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

CLI Argument: --*EvadeETW*

Advanced AMSI Evasion

Shellter Pro Plus provides the capability to inject additional code that can blind Windows' *"Antimalware Scan Interface"* (AMSI).

The Antimalware Scan Interface (AMSI) is indeed utilized by Windows Defender and other security software to scan memory buffers of a process upon request.

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

! **WARNING:** It is **not** recommended to combine our AMSI evasion methods with others that may be offered by a C2 framework or custom ones that may be copied from other sources. !

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

CLI Argument: --*EvadeAMSI/--EvadeAMSI-Enhanced*

Advanced Self-Process and Payload Threads Protection

Shellter Pro Plus provides functionality to inject code that can modify the security information of the process executing the payloads, as well as the security details of the main threads initiated by its code.

This can make can make manual debugging significantly more challenging. Certain user-mode debuggers may encounter blocked process access requests, effectively hindering their ability to analyse the execution.

Advanced Native SysCalls-Based Runtime Evasion

Shellter Pro Plus employs native syscalls to perform all these actions, effectively bypassing hooks and other monitoring mechanisms that may have been injected into the process.

Shellter Pro Plus incorporates advanced low-level techniques to obscure the usage of direct syscalls, further complicating detection by security software. By leveraging these sophisticated methods, it ensures an even greater degree of stealth, making it more challenging for monitoring tools to identify or analyse its behaviour.

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

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