University Of Piraeus

Department of Digital Systems

Postgraduate Programme "Digital Systems Security"

Master's Thesis

Analyzing the effectiveness of shellcode injectors

Chatzimangou Stamatios, MTE1636

Under the supervision of:
Dr. Christoforos Dadoyan, dadoyan@unipi.gr

Piraeus 2019

This thesis is dedicated to the people that supported me in the mission.
Special thanks to K for her patience.

Many thanks to Dadoyan as well for pushing me through this process.

Table of Contents

Table of Figures

Table of Tables

Abstract

In this thesis we analyze the effectiveness of shellcode injectors regarding their ability to bypass antivirus engines. To assist us in the process we have developed a tool written in Python 2.7 which automates the process of sample generation, analysis of the infected files, statistics calculation and presentation of results. We demonstrate the usage and results of this tool on two shellcode injectors, ROPInjector and Shellter. By generating a large sample of infected files and testing them against the online service VirusTotal we are able to demonstrate the effectiveness of each shellcode injector to hide the malicious payload as well as the effectiveness of antivirus engines to accurately detect the injected files. The output of this work is a tool that facilitates and automates this process and the highlighting the strength and weaknesses of both the shellcode injectors and the antivirus engines.

1 Motivation

The motivation for this Thesis was to study shellcode injectors and outline their strength and weaknesses as well as understand the effectiveness of antivirus engines against them. To assist us in this process we had to develop an automated way in order to massively analyze samples of carrier files and generate meaningful statistics.

In an effort to make this work usable and useful for the future and anyone how might be interested in it, a tool was developed with special care given to the design to make it as generic as possible and not shellcode injector specific as well as extensible should anyone ever need to add to it additional functionality.

To test and demonstrate the usage of the tool two shellcode injectors were selected: ROPInjector [1] and Shellter [2]. Results from the analysis of these injectors are included in this Thesis.

2 Software Documentation

In this section we provide details for the implementation of the tool, the user documentation as well as instructions on how to expand the functionality of the tool with additional plugins.

2.1 Implementation Details

The tool has been designed with the following requirements in mind:

- **Automation**: Time consuming processes like sample generation, analyzing evasion ratios and gathering data for statistical purposes should be automated.
- **Extensibility**: New functionality should be added without having to edit the existing source code.
- **Presentation:** Analysis **r**esults should be searchable and exportable and presented in a user-friendly format.
- **Execution Options**: The user should be able to configure aspects of the analysis of the samples in each run of the tool according to his needs.

To fulfill the aforementioned requirements the tool has been developed in a modular way, utilizing plugins for implementing its functionality and carrying out various analysis tasks.

As different plugins are used for different operations (generation of samples, analyzing samples, creating cumulative charts), a basic ordering mechanism has been implemented to ensure that plugins will be executed in a meaningful order. Specifically plugins fall into one of the following categories, GENERATOR, ANALYZER, PRESENTER and are executed in this order explicitly. A brief description of the categories is provided in the following table:

Table 1. Plugin types and descriptions

Plugin Type	Description
GENERATOR	Plugins of this type are responsible for generating samples give input files, run modes and payloads and are executed first.
ANALYZER	Plugins of this type are responsible for performing analysis task on the generated samples and are executed after GENERATOR plugins.
PRESENTER	Plugins of this type are responsible for creating reports, charts and calculated statistics based on analysis results and are executed last.

It is possible that a plugin may perform more than one of these operations (generation, analysis or presentation). If such is the case the plugin is given the type that allows it to be executed faster in that chain (e.g. If a plugin is generating samples and creates a report with statistics then it will be of type GENERATOR, if the plugin analyzes samples and generates a report it will be of type ANALYZER etc..). This convention ensures that multiple plugins can be chained and run in the correct order.

It is also important to note at this point that if multiple plugins of the same type are selected then they will be executed with the order that they were given in the command line.

Additionally, each plugin exposes and accepts a set of arguments allowing the user to configure its operation at run time.

The execution of the tool and the process described above are also depicted in the following flow chart while the user arguments are described in the following table:

Table 2. Arguments provided during execution of the tool

Arguments	Type	Description
--level/-l	String Choices [debug,info,warning,error,critical]	Sets the logging level for the tool and executed plugins.

--store/-s	String	A store is a directory containing the output of the plugins whether this is generated files, results from analysis or reports. Sets the store directory.
--plugin/-p	String Choices [ropinjector,shelter,virustotal,injectotal]	Sets the plugin or plugins to run.
--open-browser/-o	Boolean	Opens the report in a browser window at the end of all plugins execution.
--export/-e	Boolean	Exports the report from the store

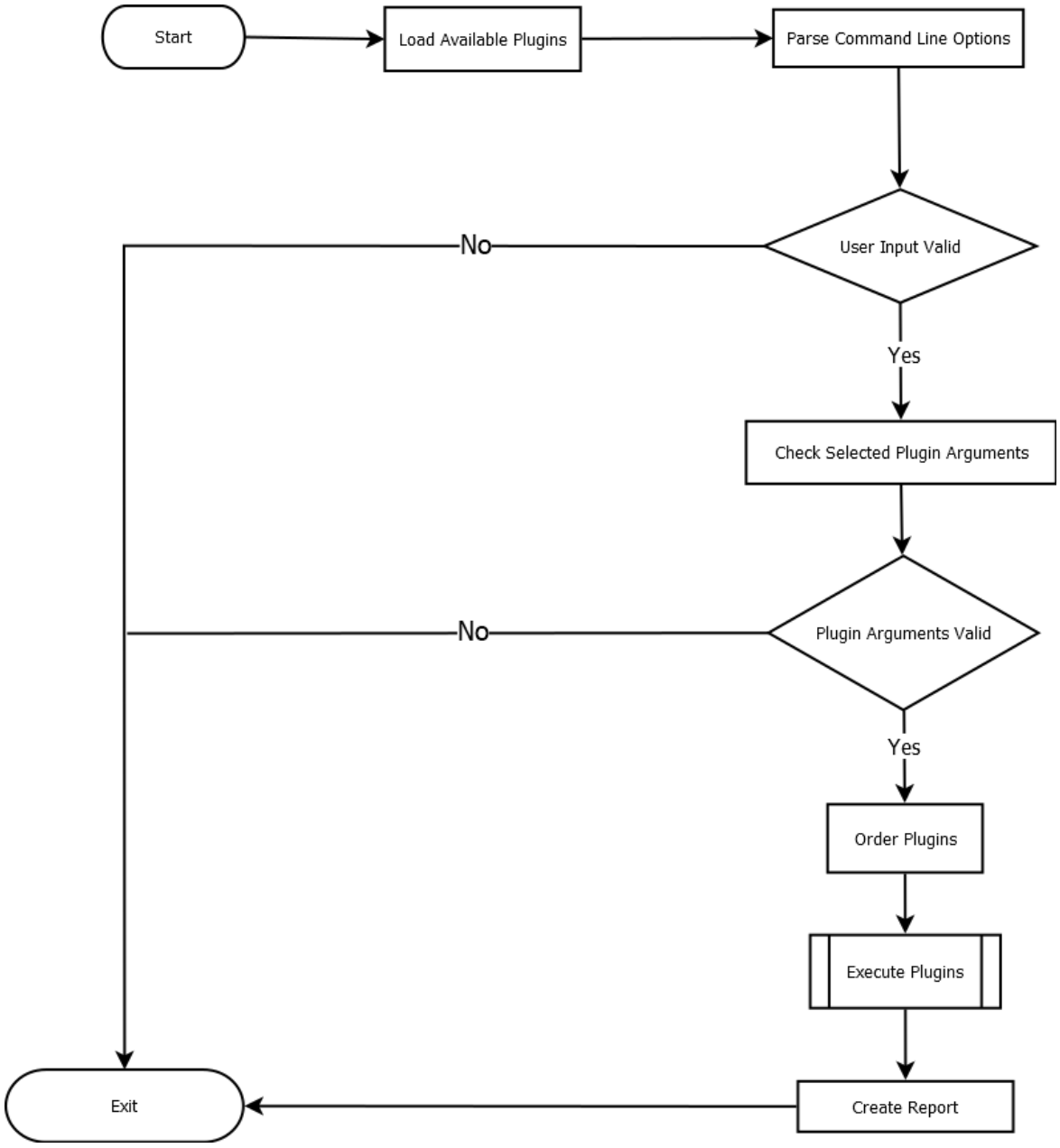


Figure 1. Flow chart of the tool execution

To perform our experiments on the shellcode injectors, a total of four plugins are developed at the time of writing. A brief description of the functionality provided by each plugin is given in the following table and a more detailed one in the sections that follow.

Table 3. Description of developed plugins

Plugin Name	Plugin Type	Description
ROPInjector	GENERATOR	This plugin is responsible for generating a carrier file samples using the ROPInjector shellcode injector.
Shellter	GENERATOR	This plugin is responsible for generating carrier file samples using the Shellter shellcode injector.
VirusTotal	ANALYZER	This plugin is responsible for analyzing the detection and evasion rates of the injected files using the VirusTotal service.
Injectotal	PRESENTER	This plugin is responsible for generating charts comparing the results of the virustotal plugin analysis for the different methods, shellcodes and engines used in the generation of the injected files.

2.2 Plugin: ROPInjector

2.2.1 Execution

The ROPInjector plugin provides all the required functionality to automate the generation of infected samples using the ROPInjector shellcode injector. Statistics provided by the ROPInjector regarding the injection are also provided as a report by this plugin. The user is able to configure the following arguments at runtime.

Table 4. ROPInjector plugin arguments

Arguments	Type	Description
--rop-directory/-ropdir	String	A directory with binaries to infect with the ROPInjector.
--rop-shellcode/-ropshell	String	A file or directory of shellcodes. If not specified the plugin will use the revshell payload calling at 127.0.0.1:4444.
--rop-args-file/- ropargsf	String	A file containing ROPInjector arguments in the following format (text norop nounroll -d5) seperated by new lines.
--rop-args/-ropargs	String	The arguments of ROPInjector to generate infected files (e.g. text entry).
--rop-version/-ropver	Integer Choices [1,2]	The version of ROPInjector to use for infection. Version 1 is the original version published in 2015. Version 2 has been enriched with more statistics and run modes.
--rop-skip/-ropskip	Boolean	Skip the generation of samples and jump to report generation. Useful for debugging reasons.

The execution of the plugin based on the arguments specified by the user is depicted in the following flow chart.

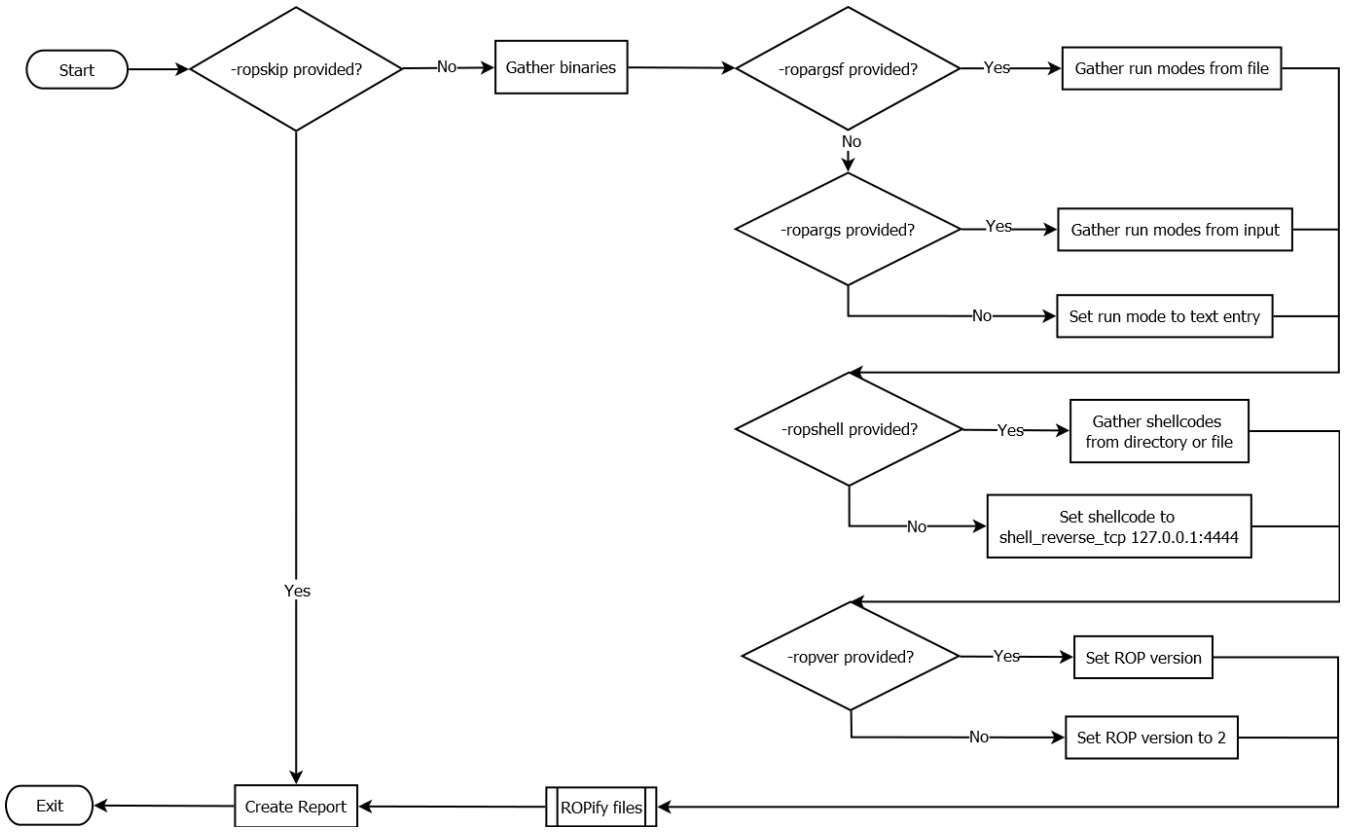


Figure 2. Flow chart of the execution of the ROPInjector plugin

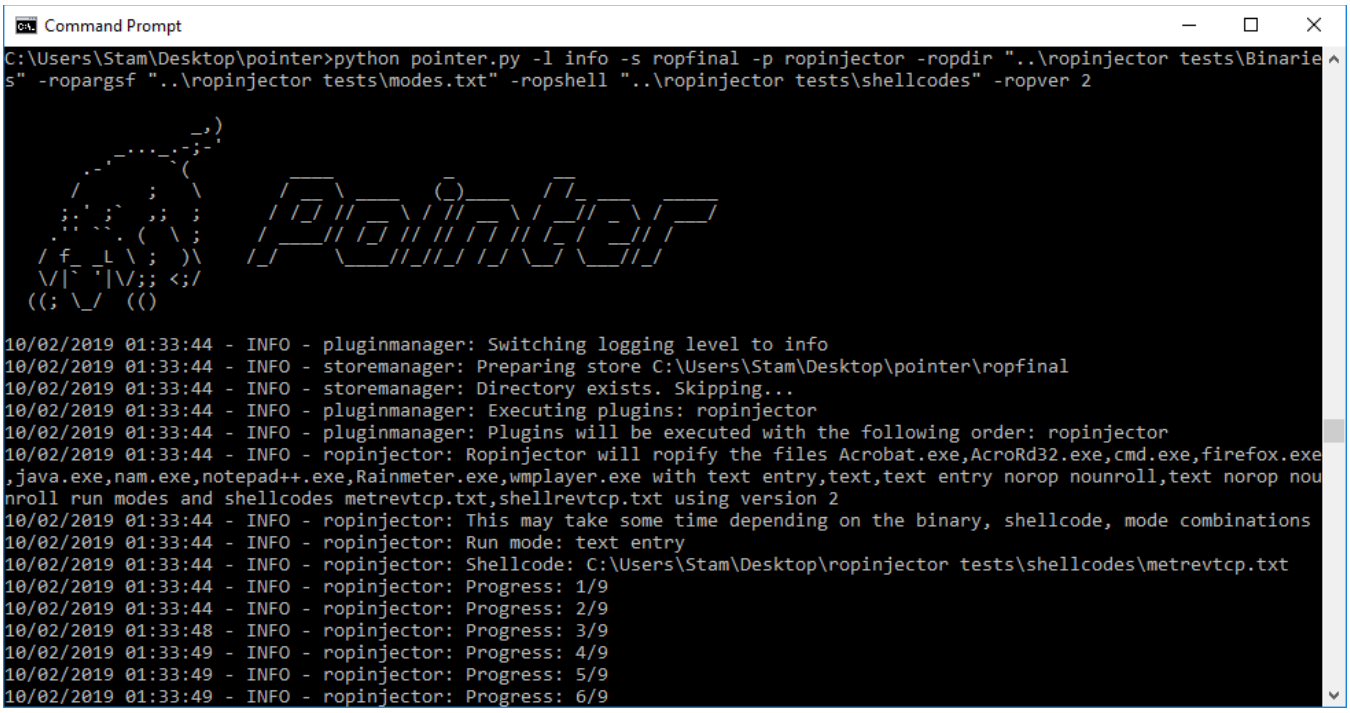


Figure 3. Execution of the ROPInjector plugin in command line

2.2.2 Report

The plugin will generate 2 reports with statistics, information for each one of the injected files and comparison graphs. The first report is called ROPInjector and includes statistics and information from the injection of each file. Details regarding the information is provided can be found in the next table.

Table 5. ROPInjector report information and statistics

Statistic / Information	Description
ID	The ID of the file injected. The ID has the format filename / shellcode / injection method.
PE Size	Initial size of the PE file in Kbytes.
Shellcode Size	Shellcode size in bytes.
Patch Size	Patch size in bytes.
Gadgets in PE	Number of candidate gadgets identified in the PE.
Instructions replaced with gadgets	Number of instructions replaced by ROP gadgets.
Instructions non ropable	Number of Instructions that could not be transformed to ROP as they are not supported by the tool.
Instructions replaced by injected gadgets	Number of instructions replaced by injected gadgets.
Gadgets Injected	Number of gadgets that were injected.
Gadgets injected with pseudofunctions	Number of instructions replaced by injected gadgets of instructions replaced by injected using a pseudofunction.

Gadgets injected with epilogue ext	Number of instructions replaced by injected gadgets using an existing function epilogue extension.
.text ext	Number of times that the text section was extended.
Gadgets Not Injected	Number of gadgets that exist in the PE and were used by the ROPInjector.
(%) of Gadgets Injected (%)	Percentage of gadgets injected as opposed to the ones that were used from the original PE.
(%) of Gadgets Used (%)	Percentage of gadgets used from the candidate gadgets identified in the PE.
Gadget Segments	Number of gadget segments.
Entry	Whether access is given to the shellcode during entry (run first) or during exit (run last).
Delay	The delay the shellcode sleeps before it runs in seconds.
No Rop	Whether the original shellcode is transformed to ROP or is patched intact.
No Unroll	Whether shellcode has been converted to ROP.
getPC	Whether getPC constructs are replaced in the shellcode.
Inject Gadgets	Whether missing gadgets were injected.
Hide Certificate	Whether the certificate was hidden or deleted.

A sample screenshot from the generated report is provided below. All results in the table are searchable, sortable and exportable.

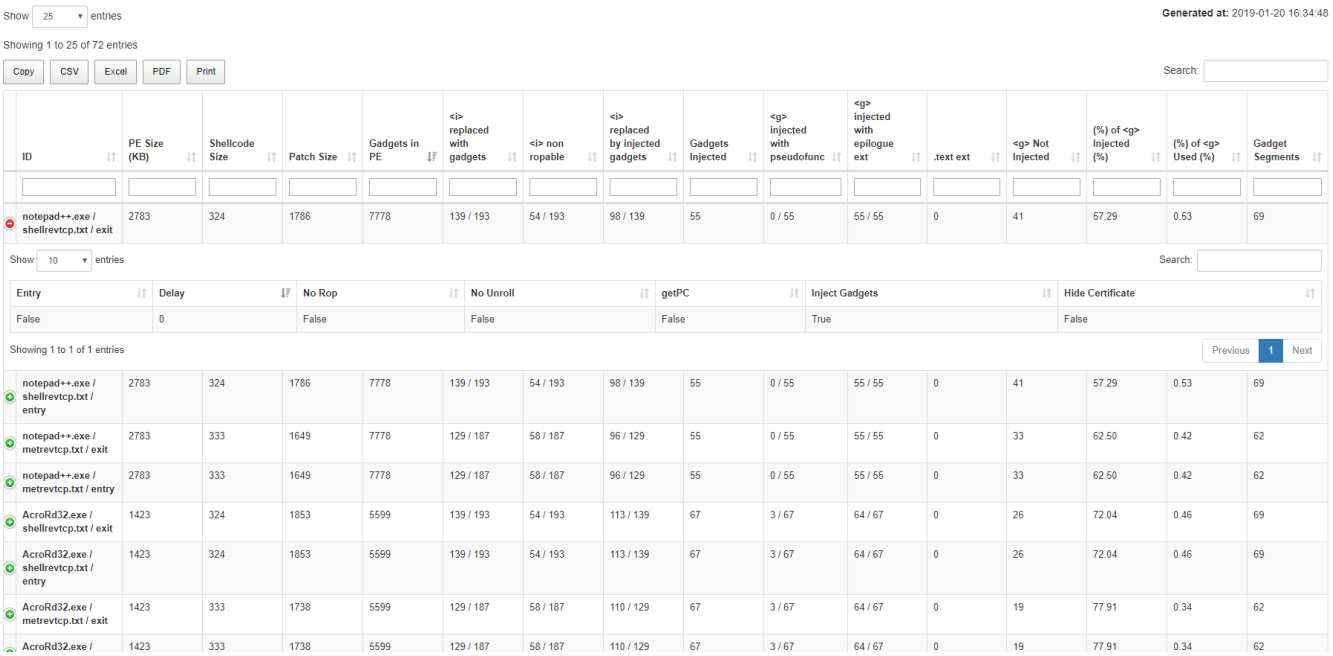


Figure 4. Statistics report from ROPInjector plugin execution

The second report is called **ROPInjector Graphs** and includes 3 comparison charts with the following information:

Table 6. Comparison bar chart information from the ROPInjector Graphs report

Chart Title	Description
Gadgets Injected vs Gadgets not Injected	This chart compares the gadgets that were injected in the PE versus the ones that were found in the PE and where used by ROPInjector.
Gadgets Injected with Pseudofunctions vs Gadgets Injected with Epilogue Extensions	This chart compares the gadgets that were injected by inserting a pseudofuntion in the PE versus the ones that were injected by extending the epilogue of existing functions found in the PE.
Candidate Gadgets in PE vs Gadgets Used	This chart compares the candidate gadgets identified in the PE versus the ones that were actually used for the injection.

A sample screenshot from each chart in this report can be found below.

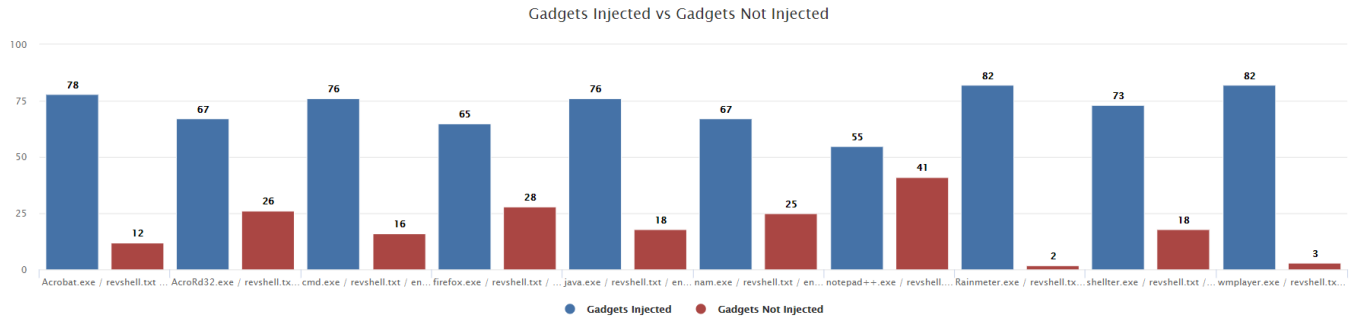


Figure 5. Gadgets Injected vs Gadgets not Injected comparison bar chart

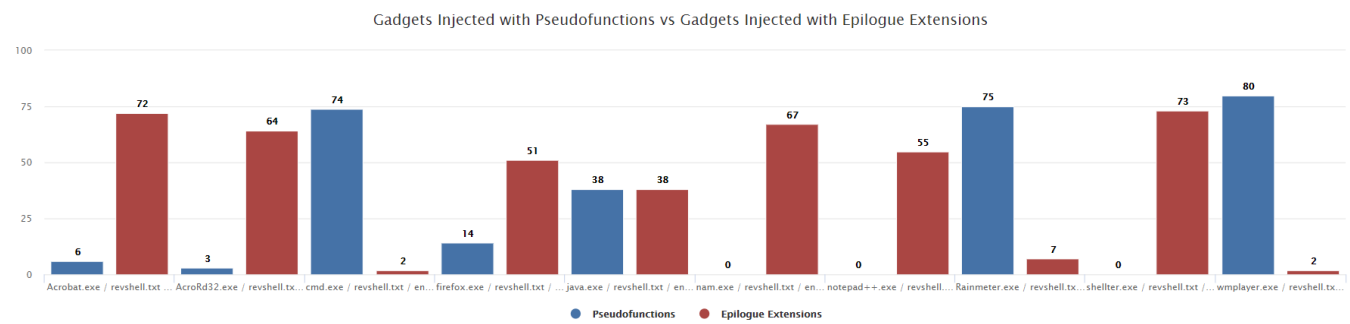


Figure 6. Gadgets Injected with Pseudofunctions vs Gadgets Injected with Epilogue Extensions comparison bar chart

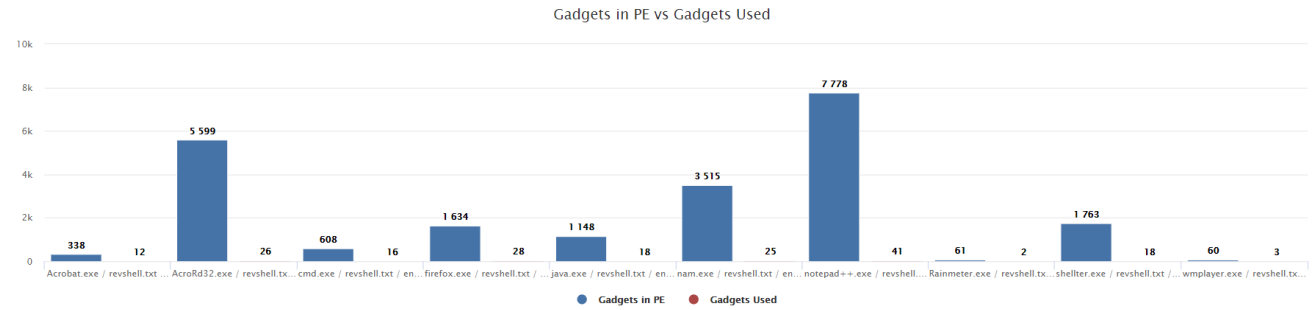


Figure 7. Gadgets in PE vs Gadgets Used comparison bar chart

2.3 Plugin: Shellter

2.3.1 Execution

The Shellter plugin provides all the required functionality to automate the generation of infected samples using the Shellter shellcode injector.

Table 7. Shellter plugin arguments

Arguments	Type	Description
--shellter-directory/-stdir	String	A directory with binaries to infect with Shellter.
--shellter-args-file/-stargsf	String	A file containing Shellter arguments in the following format (-a -s -p meterpreter_reverse_tcp --lhost 192.168.233.100 --port 4444) seperated by new lines.
--shellter-args/-stargs	String	The arguments of Shellter to generate infected files (e.g. -a -p shell_reverse_tcp --lhost 192.168.233.100 --port 4444).
--shellter-skip/-stskip	Boolean	Skip the generation of samples and jump to report generation. Useful for debugging reasons.

The execution of the plugin based on the arguments specified by the user is depicted in the following flow chart.

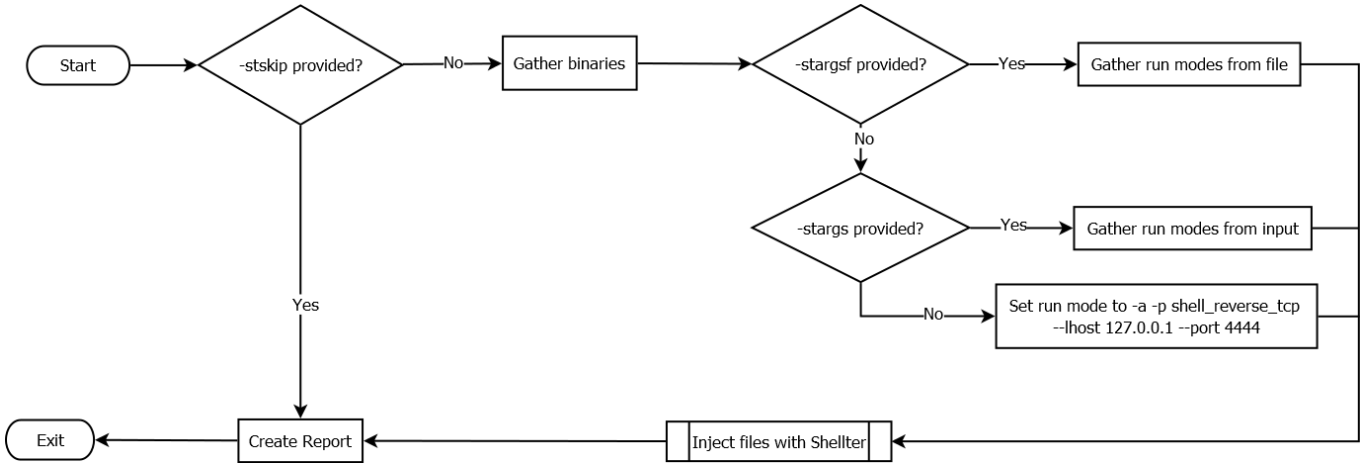


Figure 8. Flow chart of the execution of the Shellter plugin

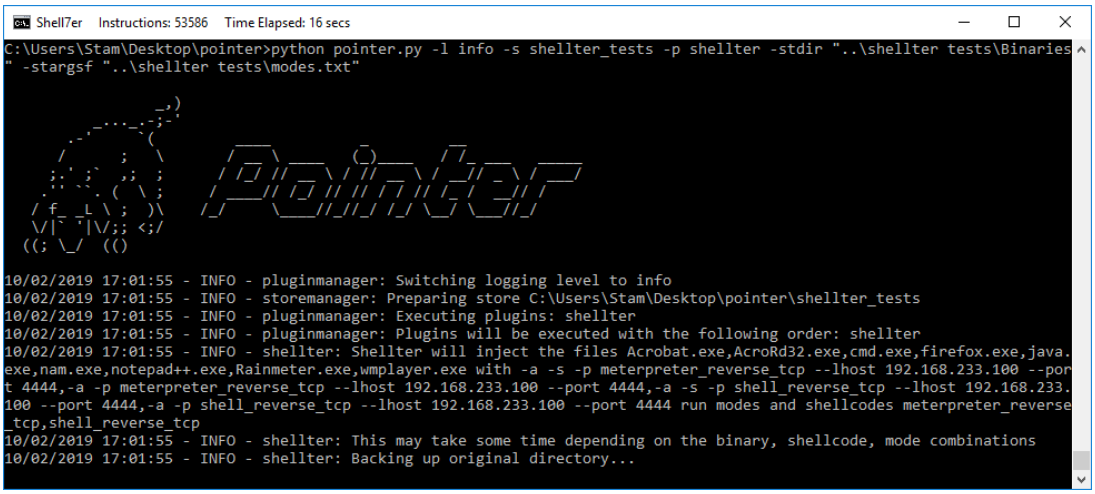


Figure 9. Execution of the Shellter plugin in command line

2.3.2 Report

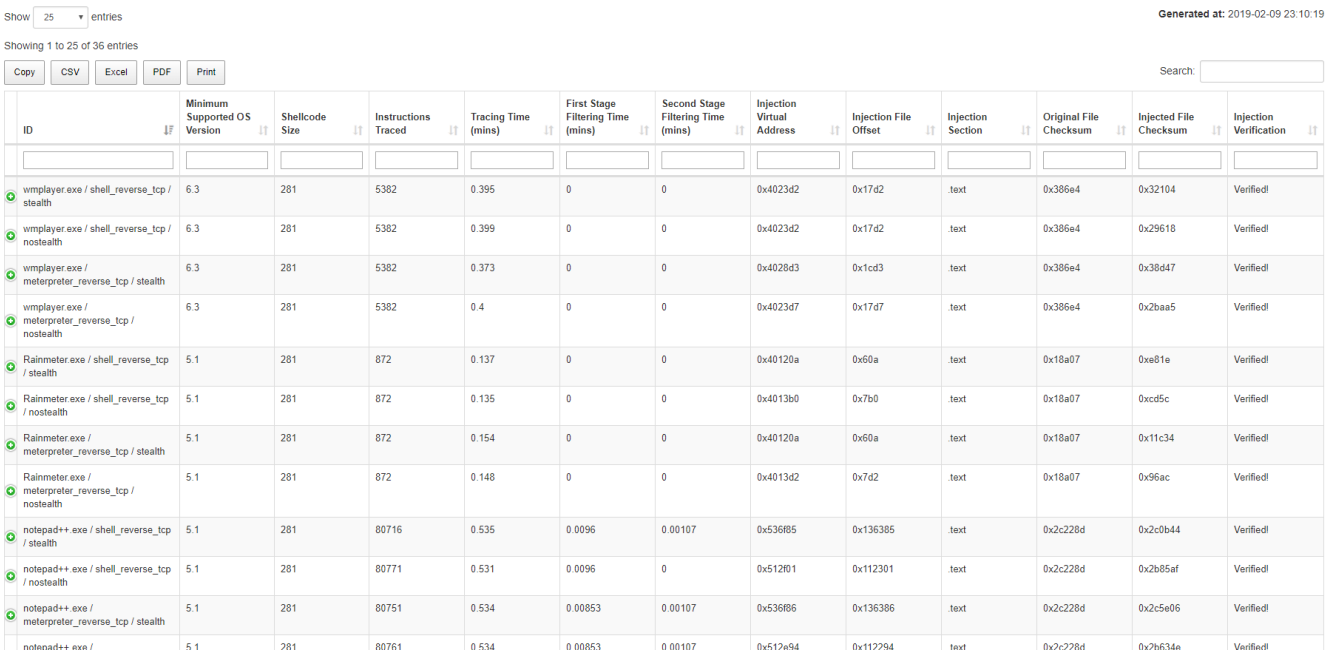
The plugin generates one report with information from the output of the injection with Shellter. The report is called Shellter and has the following information.

Table 8. Shellter report information

Information	Description
ID	The ID of the injected file. The ID has the format filename / shellcode / injection method.
Minimum Supported OS Version	The minimum required Windows version for the target application to run. This information is taken directly from the PE header and might be not always accurate.
Shellcode Size	The size of the payload that was injected in the file.
Instructions Traced	The number of instructions traced by Shellter. In Auto Mode, Shellter will trace a random number of instructions for a maximum time of approximately 30 seconds in native Windows hosts and for 60 seconds when used in Wine.
Tracing Time	The time that shellter was tracing instructions in minutes.
First Stage Filtering Time	Time taken for first stage filtering to complete.
Second Stage Filtering Time	Time taken for second stage filtering to complete.
Injection Virtual Address	The virtual address of the first instruction of the injected code
Injection File Offset	The offset of the first instruction of the injected code.
Original File Checksum	The checksum of the file before the injection.
Injected File Checksum	The checksum of the file after the injection.
Injection Verification	Whether the injection was successful.
Packed	Whether the file is packed.

Elimination Status	Whether data were eliminated on the injected file.
Elimination Data	Type of data eliminated from the injected file.
Reflective Loader	Whether a reflective loader is used.
Encode Payload Handling	Whether encode-payload handling is enabled or disabled.
Handler Type	The handler type selected for the injection.

A sample screenshot from the generated report is provided below.



Show 25 entries

Generated at: 2019-02-09 23:10:19

Showing 1 to 25 of 36 entries

Copy CSV Excel PDF Print

Search:

ID	Minimum Supported OS Version	Shellcode Size	Instructions Traced	Tracing Time (mins)	First Stage Filtering Time (mins)	Second Stage Filtering Time (mins)	Injection Virtual Address	Injection File Offset	Injection Section	Original File Checksum	Injected File Checksum	Injection Verification
wmplayer.exe / shell_reverse_tcp / stealth	6.3	281	5382	0.395	0	0	0x4023d2	0x17d2	.text	0x386e4	0x32104	Verified!
wmplayer.exe / shell_reverse_tcp / nostealth	6.3	281	5382	0.399	0	0	0x4023d2	0x17d2	.text	0x386e4	0x29618	Verified!
wmplayer.exe / meterpreter_reverse_tcp / stealth	6.3	281	5382	0.373	0	0	0x4028d3	0x1cd3	.text	0x386e4	0x38d47	Verified!
wmplayer.exe / meterpreter_reverse_tcp / nostealth	6.3	281	5382	0.4	0	0	0x4023d7	0x17d7	.text	0x386e4	0x2baa5	Verified!
Rainmeter.exe / shell_reverse_tcp / stealth	5.1	281	872	0.137	0	0	0x40120a	0x60a	.text	0x18a07	0xe81e	Verified!
Rainmeter.exe / shell_reverse_tcp / nostealth	5.1	281	872	0.135	0	0	0x4013b0	0x7b0	.text	0x18a07	0xcd5c	Verified!
Rainmeter.exe / meterpreter_reverse_tcp / stealth	5.1	281	872	0.154	0	0	0x40120a	0x60a	.text	0x18a07	0x11c34	Verified!
Rainmeter.exe / meterpreter_reverse_tcp / nostealth	5.1	281	872	0.148	0	0	0x4013d2	0x7d2	.text	0x18a07	0x96ac	Verified!
notepad++.exe / shell_reverse_tcp / stealth	5.1	281	80716	0.535	0.0096	0.00107	0x536f85	0x136385	.text	0x2c228d	0x2c0b44	Verified!
notepad++.exe / shell_reverse_tcp / nostealth	5.1	281	80771	0.531	0.0096	0	0x512f01	0x112301	.text	0x2c228d	0x2b85af	Verified!
notepad++.exe / meterpreter_reverse_tcp / stealth	5.1	281	80751	0.534	0.00853	0.00107	0x536f86	0x136386	.text	0x2c228d	0x2c5e06	Verified!
notepad++.exe /	5.1	281	80761	0.534	0.00853	0.00107	0x512e94	0x112294	.text	0x2c228d	0x2b634e	Verified!

Figure 10. Shellter plugin report

2.4 Plugin: VirusTotal

2.4.1 Execution

The VirusTotal plugin provides the functionality required to massively submit files to the VirusTotal online service [3] for analysis and retrieve the results. The plugin will also generate 4 reports based on the most effective Engines, most common Signatures among the analyzed sample and most detected files.

Table 9. VirusTotal plugin arguments

Arguments	Type	Description
--virustotal-key/-vtkey	String	The VirusTotal API Key required to submit files.
--virustotal-limit/-vtlim	Integer	The limit of requests per minute.
--virustotal-dir/-vtdir	String	The directory of files to analyze.
--virustotal-recursion/-vtrec	Boolean	Whether the files will be detected recursively in the directory.
--virustotal-file-types/-vtfmt	String	The extension of the files that will be uploaded.
--virustotal-noscan/vtno	Boolean	Fetch the VirusTotal reports for files already submitted. For the rest of the files skip analysis.
--virustotal-mixscan/-vtmix	Boolean	Fetch the VirusTotal reports for files already submitted. For the rest of the files upload and fetch the report.
--virustotal-new/-vtnew	Boolean	Scan only files for which reports do not exist in the store.
--virustotal-immediate/-vtimm	Integer	Send x requests for and then query for reports. This mode is useful when scanning a large dataset with a limit in the requests per minute and ensures that you will retrieve results as fast as possible. This value should never be greater than vtlim.
--virustotal-skip/-vtskip	Boolean	Skip plugin analysis and jump to report generation. Useful for debugging reasons.

The execution of the plugin based on the arguments specified by the user is depicted in the following flow chart.

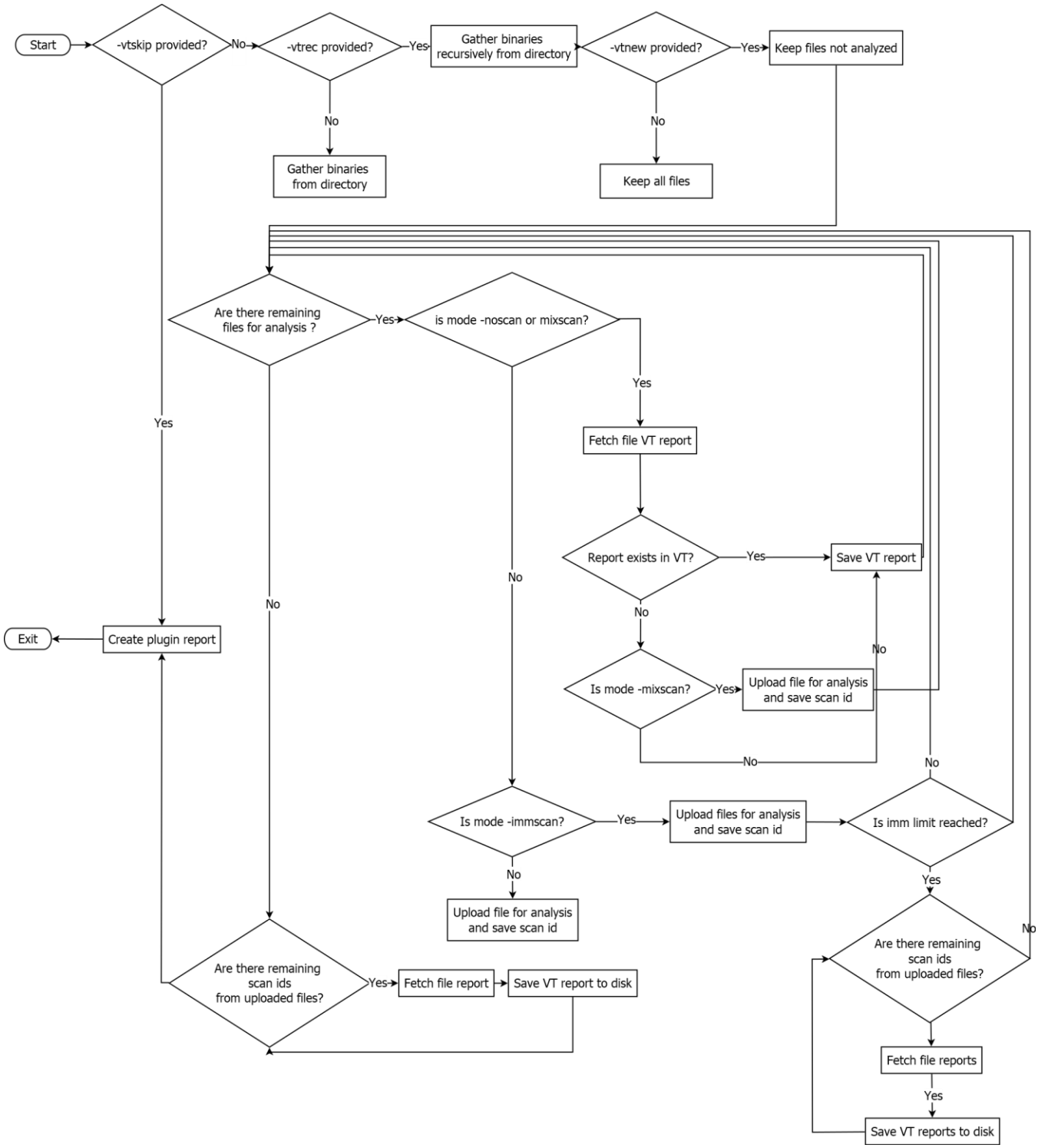


Figure 11. Flow chart of the execution of VirusTotal plugin

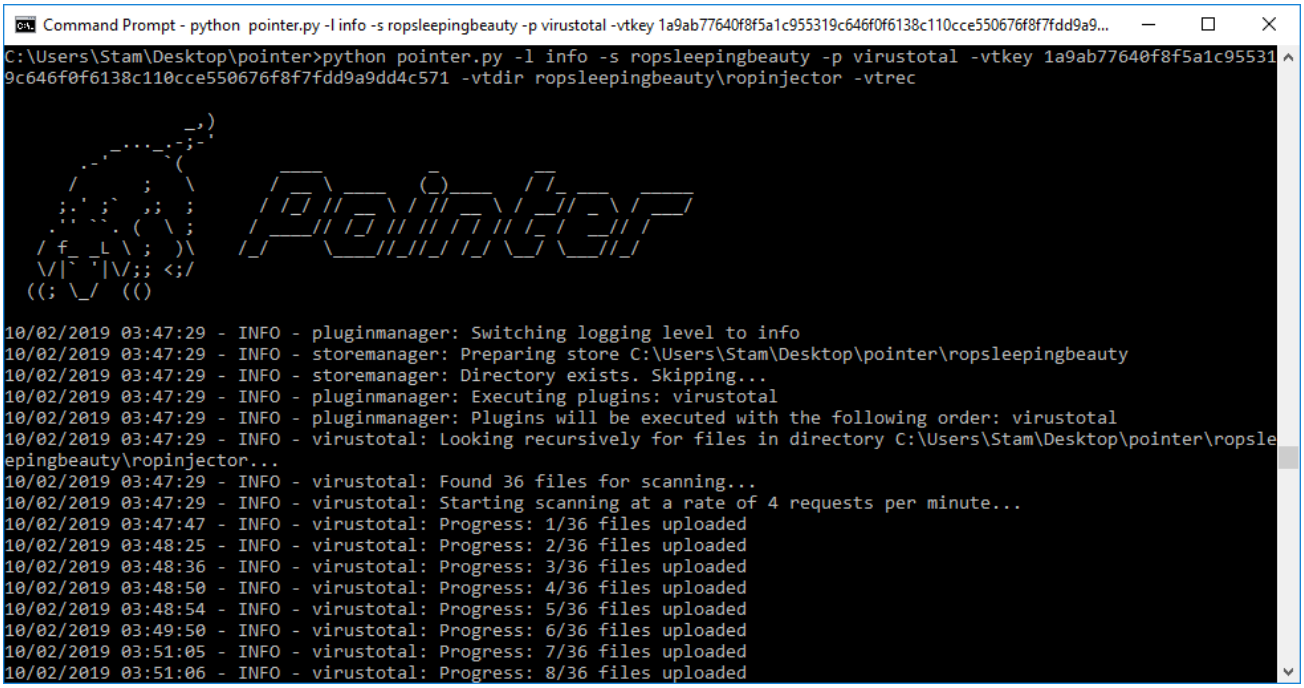


Figure 12. Execution of the virustotal plugin in command line

2.4.2 Report

Samples screenshots from the generated reports are provided below. The tool generates in total 4 reports.

The first report is the VirusTotal Dashboard which allows easy navigation between the engines, signatures and detected files from the analyzed samples. A sample screenshot is provided below.

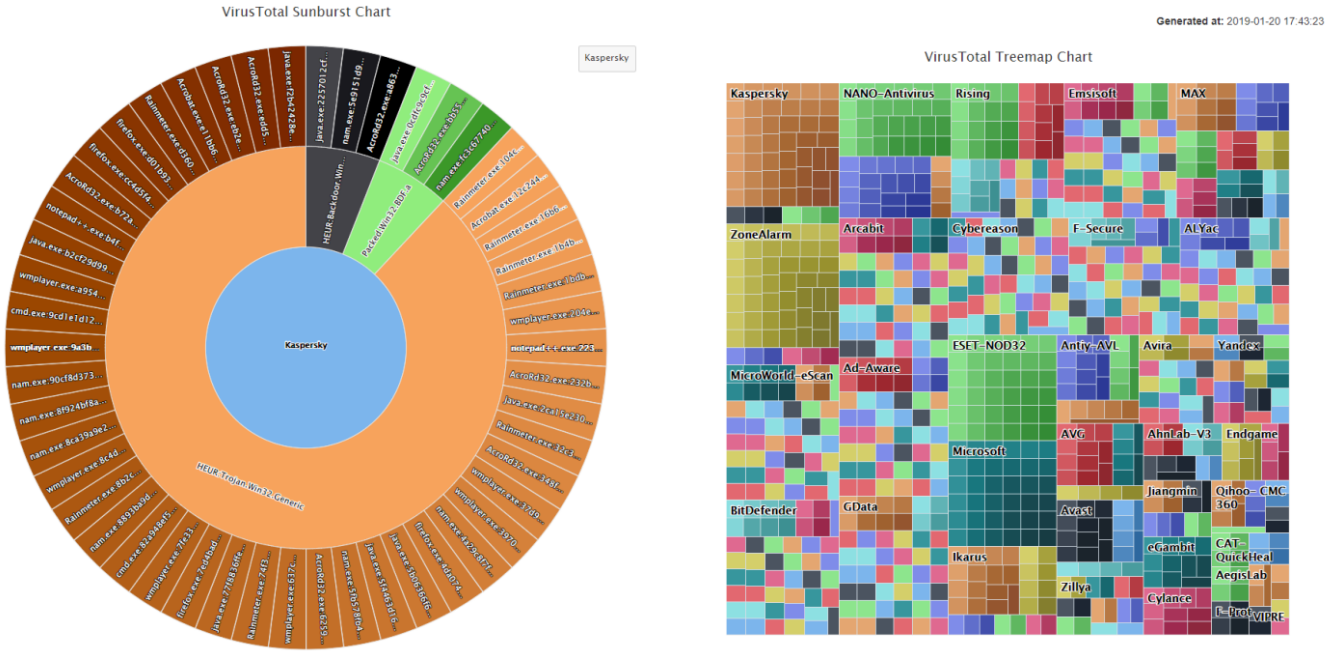


Figure 13. VirusTotal sunburst and treemap charts generated from the VirusTotal plugin

The second report is called VirusTotal Engines and provides statistics for the effectiveness of each Antivirus Engine against the analyzed sample. Information displayed on the report is provided below. Additionally 3 charts are provided: A semi circle donut displaying the number of engines that managed to detect at least one file vs the number of engines that had no detections, a spider chart displaying a keyword analysis on the signatures that were triggered for the analyzed sample and a word cloud chart displaying the most prominent engines (meaning the ones with the most detections).

Table 10. Statistics and information in VirusTotal Engines report

Statistic / Information	Description
Antivirus Engines	The name of the antivirus engine.
Detections	The absolute value of the number of detections performed by the antivirus engine.
Detection Ratio of Total Files (%)	The percentage of files detected from the antivirus engine from the total analyzed sample.
Evasion Ratio of Total Files (%)	The percentage of files that evaded detection from the antivirus engine from the total analyzed sample.
Unique Signatures	The same signature can be used to detect multiple infected files. This column will display the number of **unique** signatures where used by the antivirus engine to make the detection.
Spider Chart	A spider chart is generated for every engine. The plugin will perform a keyword analysis on the signatures and display a spiderchart with the most common keywords for each engine.

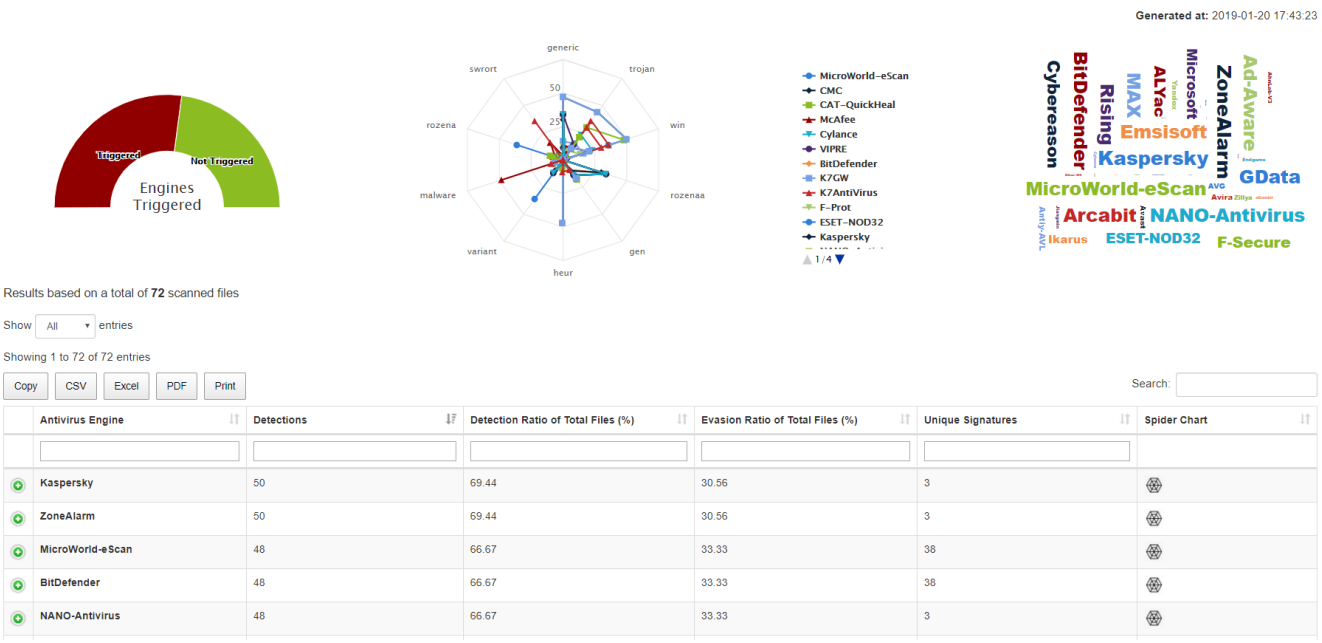


Figure 14. VirusTotal Engines report statistics and information

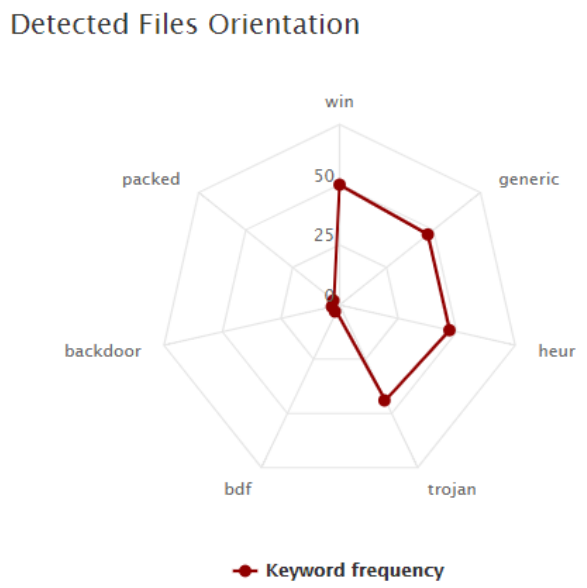


Figure 15. Example of VirusTotal Engines report Engine spider chart

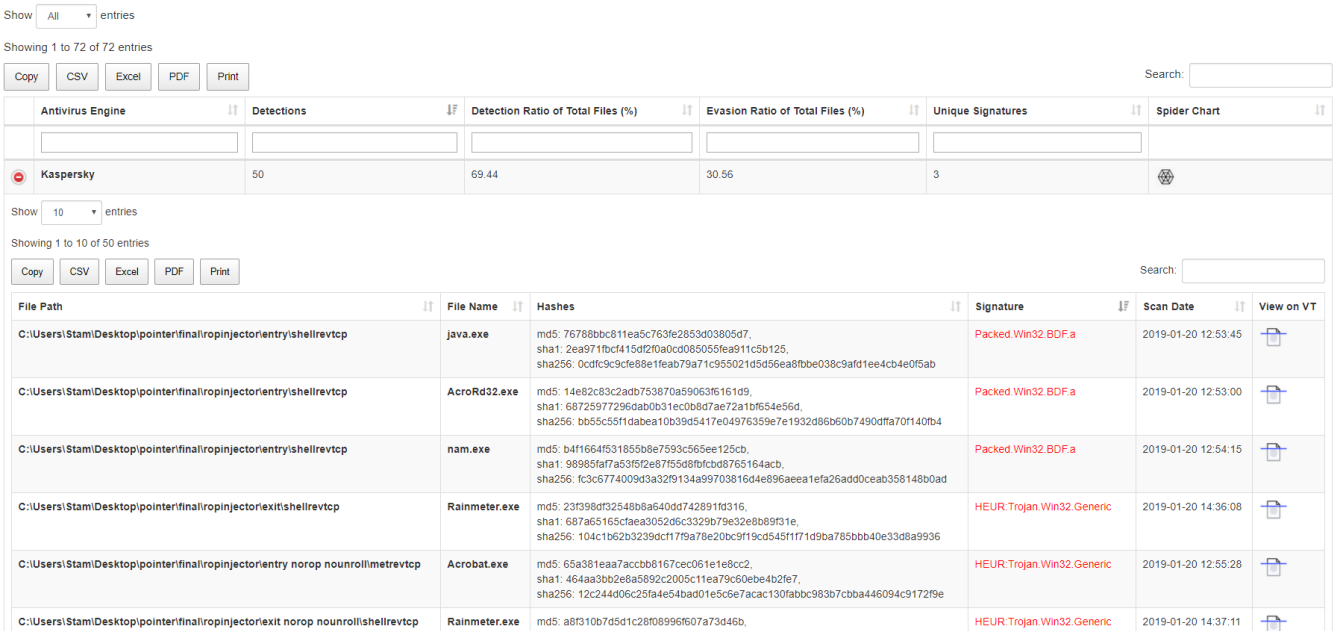


Figure 16. Expanded view of a specific antivirus engine

The third report is the VirusTotal Signatures report which includes information on the triggered signatures. This particular report is useful for hinting what is detected by the antivirus engines on the analyzed samples. The following statistics and information are provided in this report.

Table 11. Statistics and information in VirusTotal Signatures report

Statistic / Information	Description
Signature Name	The name of the signature that made the detection.
Total File Detections	The absolute value of the number of detections performed by this signature.

Unique File Names	The number of unique file names that were detected. (Useful when analyzing samples generated using a different method or payload).
Appearance in Engines	The number of engines that this signature appears in.

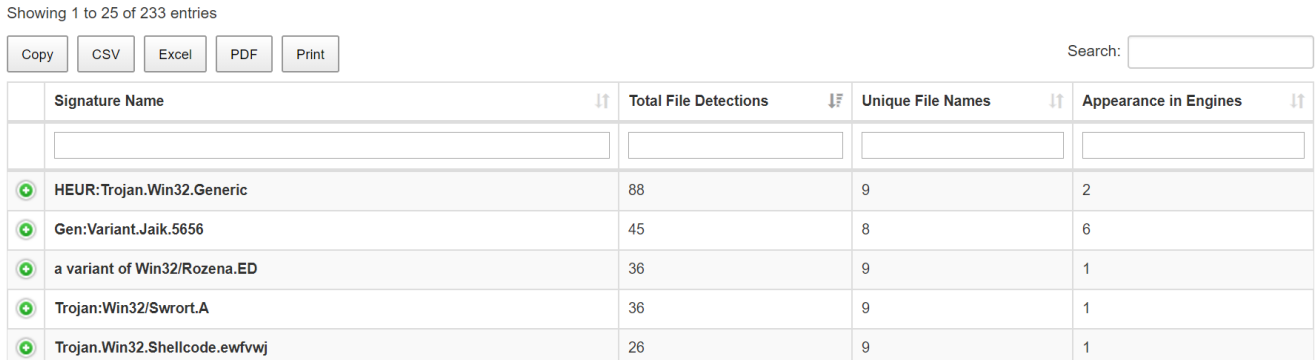


Figure 17. VirusTotal Signatures report statistics and information

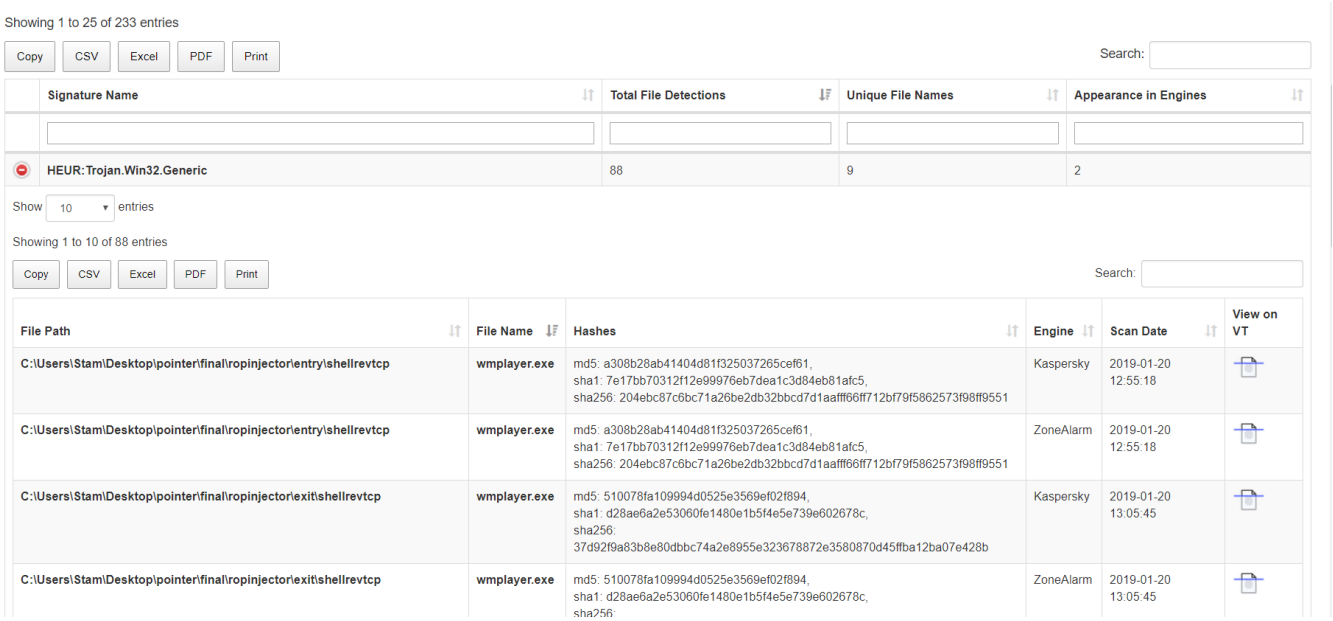


Figure 18. Expanded view of a specific signature in VirusTotal Signatures report

Additionally the report includes 3 charts: a pie chart displaying the top 10 most frequent signatures, a word cloud with the most prominent keywords in the signatures and a searchable and exportable table with the most frequent keyword appearances.

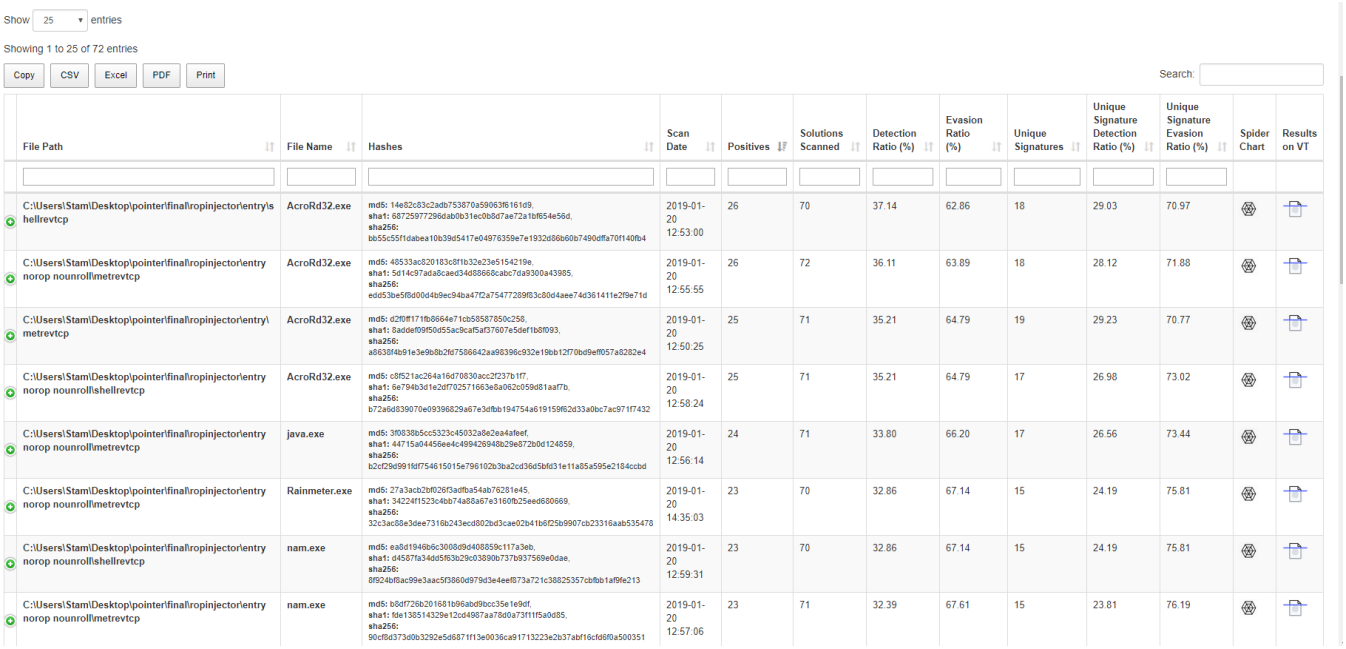
Figure 19. Charts in the VirusTotal Signatures report

Finally, the last report is the VirusTotal report which includes information on the engines and signatures that detected each file as well as some cumulative stats on the detection of the total sample that was analyzed. Specifically the report includes the following statistics.

Table 12. Statistics and information in VirusTotal report

Statistic / Information	Description
File Path	The path of the file that was analyzed.
File Name	The name of the file that was analyzed..
Hashes	The MD5, SHA1, SH256 hash of the file that was analyzed.
Scan Date	The date that that the file was scanned by the VT online service.
Positives	The number of Antivirus engines that detected the file.
Solutions Scanned	The number of Antivirus solutions that scanned the file.
Detection Ratio (%)	The percentage of positive detections for the file.
Evasion Ratio (%)	The percentage of no detections for the file.
Unique Signatures	The number of unique signatures that detected the file.
Unique Signature Detection Ratio (%)	The percentage of positive detections for the file based on the unique signatures that detected it. It is not uncommon for different antivirus solutions to use the same database of signatures. This metric assumes that if a file was detected with the same signature from different engines then these detections will be counted as 1 therefore decreasing the detection and increasing the evasion rates.
Unique Signature Evasion Ratio (%)	The percentage of no detections for the file based on the unique signatures. It is not uncommon for different antivirus solutions to use the same database of signatures. This metric assumes that if a file was detected with the same signature from different engines then these detections will be counted as 1 therefore decreasing the detection and increasing the evasion rates.
Spider Chart	A spider chart is generated for every file. The plugin will perform a keyword analysis on the signatures and display a spider chart with the most common keywords for each file.
Results on VT	A hyperlink to the VirusTotal scan results for the specific file.

Screenshots from the report are provided below.



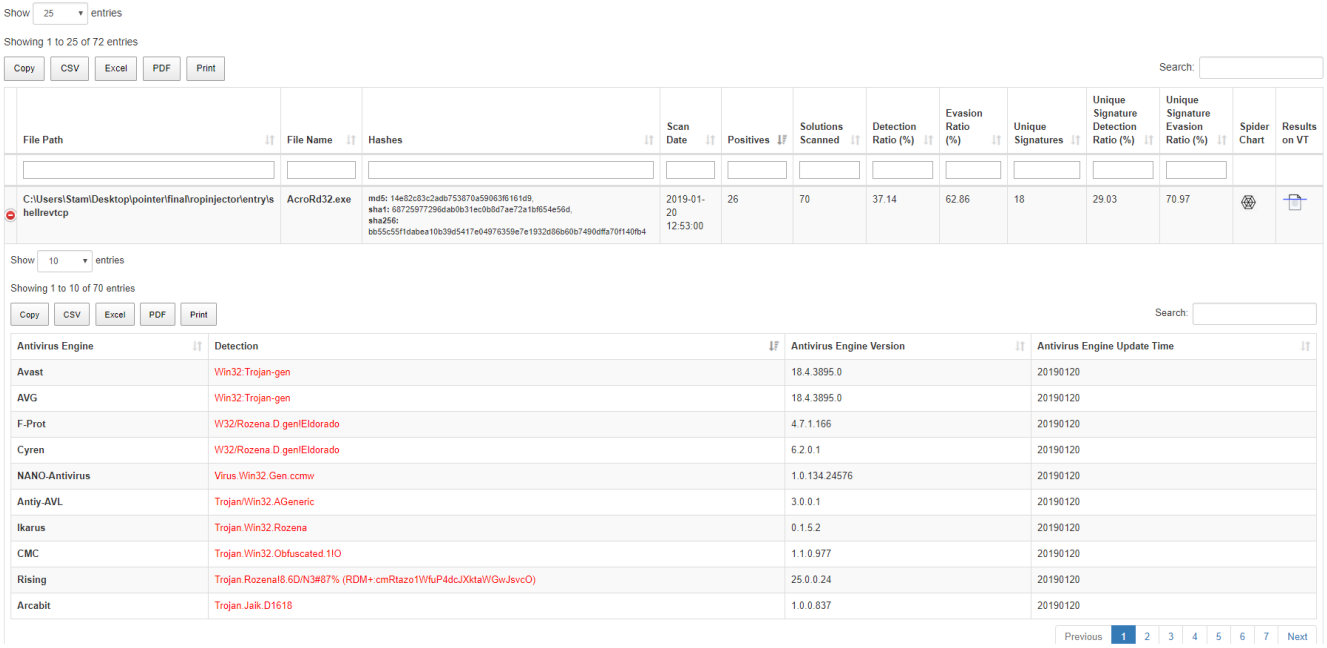
Show 25 entries

Showing 1 to 25 of 72 entries

Copy CSV Excel PDF Print Search:

File Path	File Name	Hashes	Scan Date	Positives	Solutions Scanned	Detection Ratio (%)	Evasion Ratio (%)	Unique Signatures	Unique Signature Detection Ratio (%)	Unique Signature Evasion Ratio (%)	Spider Chart	Results on VT
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry\shellrevtcp	AcroRd32.exe	md5: 14e82c83c2adb753870a59063f6161d9, sha1: 6872597726 9dab0b31ec0b8d7ae72a1bf654e56d, sha256: bb55c55f1dabea10b39d5417e04976359e7e1932d86b60b7490dffa70f140fb4	2019-01-20 12:53:00	26	70	37.14	62.86	18	29.03	70.97		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\metrevtcp	AcroRd32.exe	md5: 48533ac820183c8f1b32e23e5154219e, sha1: 5d14c97ada8caed34d88668cabc7da9300a43985, sha256: edd53be5f8d00d4b9ec94ba47f2a7547728 9f83c80d4aee74d36141 1e2f9e71d	2019-01-20 12:55:55	26	72	36.11	63.89	18	28.12	71.88		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry\metrevtcp	AcroRd32.exe	md5: d2f0ff171fb8664e71cb58587850c258, sha1: 3addef09f50d55ac9caf5af37607e5def1b8f093, sha256: a8638f4b91e3e9b8b2fd758664 2aa98396c932e19bb12f70bd9eff057a8282e4	2019-01-20 12:50:25	25	71	35.21	64.79	19	29.23	70.77		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\shellrevtcp	AcroRd32.exe	md5: c8f521ac264a16d70830acc2f237b1f7, sha1: 6e794b3d1e2df702571663e8a962c059d81aaf7b, sha256: b72a6d839070e09396829a67e3dfbb194754a619159f62d33a0bc7ac971f7432	2019-01-20 12:58:24	25	71	35.21	64.79	17	26.98	73.02		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\metrevtcp	java.exe	md5: 3f0838b5cc5323c45032a8e2ea4afeef, sha1: 44715a04456ee4c499426948b29e872b0d124859, sha256: b2cf29d991fdf75461 5015e796102b3ba2cd36d5bfd31e11a85a595e2184ccbd	2019-01-20 12:56:14	24	71	33.80	66.20	17	26.56	73.44		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\metrevtcp	Rainmeter.exe	md5: 27a3acb2bf026f3adfba54ab76281e45, sha1: 34224f1523c4bb74a68a67e3160fb25eed680669, sha256: 32c3ac88e3dee7316b243ecd802bd3cae02b41b6f25b9907cb23316aab535478	2019-01-20 14:35:03	23	70	32.86	67.14	15	24.19	75.81		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\shellrevtcp	nam.exe	md5: ea8d1946b6c3008d9d408859c117a3eb, sha1: d4587fa34dd5f63b29c03890b737b937569e0dae, sha256: 8f924bf8ac99e3aac5f3860d979d3e4eef873a721c38825357cbfbb1af9fe213	2019-01-20 12:59:31	23	70	32.86	67.14	15	24.19	75.81		
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry norop nounroll\metrevtcp	nam.exe	md5: b5df726b201681b96abd9bcc35e1e9df, sha1: fde138514329e12cd4987aa78d0a73f11f5a0d85, sha256: 90cf8d373d0b3292e5d6871ff13e0036ca91713223e2b37abf16cfd6f0a500351	2019-01-20 12:57:06	23	71	32.39	67.61	15	23.81	76.19		

Figure 20. VirusTotal report statistics and information



Show 25 entries

Showing 1 to 25 of 72 entries

Copy CSV Excel PDF Print Search:

File Path	File Name	Hashes	Scan Date	Positives	Solutions Scanned	Detection Ratio (%)	Evasion Ratio (%)	Unique Signatures	Unique Signature Detection Ratio (%)	Unique Signature Evasion Ratio (%)	Spider Chart	Results on VT
C:\Users\Stam\Desktop\pointer\final\ropinjector\entry\shellrevtcp	AcroRd32.exe	md5: 14e82c83c2adb753870a59063f6161d9, sha1: 6872597726 9dab0b31ec0b8d7ae72a1bf654e56d, sha256: bb55c55f1dabea10b39d5417e04976359e7e1932d86b60b7490dffa70f140fb4	2019-01-20 12:53:00	26	70	37.14	62.86	18	29.03	70.97		

Show 10 entries

Showing 1 to 10 of 70 entries

Copy CSV Excel PDF Print Search:

Antivirus Engine	Detection	Antivirus Engine Version	Antivirus Engine Update Time
Avast	Win32:Trojan-gen	18.4.3895.0	20190120
AVG	Win32:Trojan-gen	18.4.3895.0	20190120
F-Prot	W32/Rozena.D.gen!Eldorado	4.7.1.166	20190120
Cyren	W32/Rozena.D.gen!Eldorado	6.2.0.1	20190120
NANO-Antivirus	Virus.Win32.Gen.ccmw	1.0.134.24576	20190120
Antiy-AVL	Trojan/Win32.AGeneric	3.0.0.1	20190120
Ikarus	Trojan.Win32.Rozena	0.1.5.2	20190120
CMC	Trojan.Win32.Obfuscated.1!O	1.1.0.977	20190120
Rising	Trojan.Rozena!8.6D/N3#87% (RDM+:cmRtazo1WfuP4dcJXktaWGwJsvcO)	25.0.0.24	20190120
Arcabit	Trojan.Jaik.D1618	1.0.0.837	20190120

Previous 1 2 3 4 5 6 7 Next

Figure 21. Expanded view of a specific file in VirusTotal report

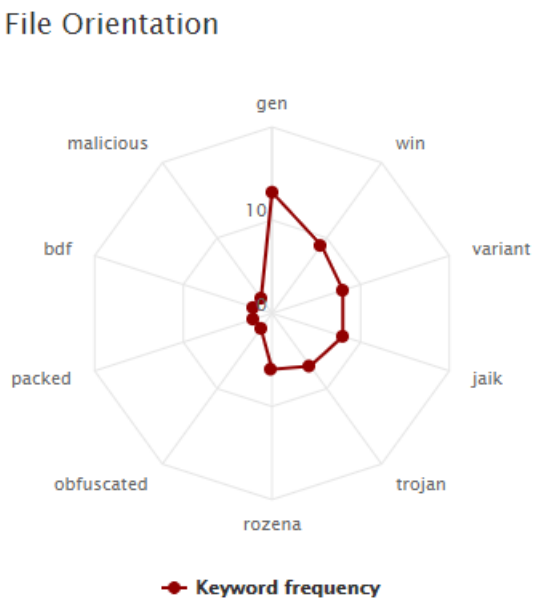


Figure 22. Example of VirusTotal report spider chart

The VirusTotal report also includes 4 charts: a semi circle donut with the number of files that have been detected by at least 1 antivirus solution vs the ones that were not detected at all, a spider chart with the top 10 most effective Antivirus Engines, a spider chart with the top 10 most frequent signatures and a spider chart with the top 10 most frequent signature keywords.

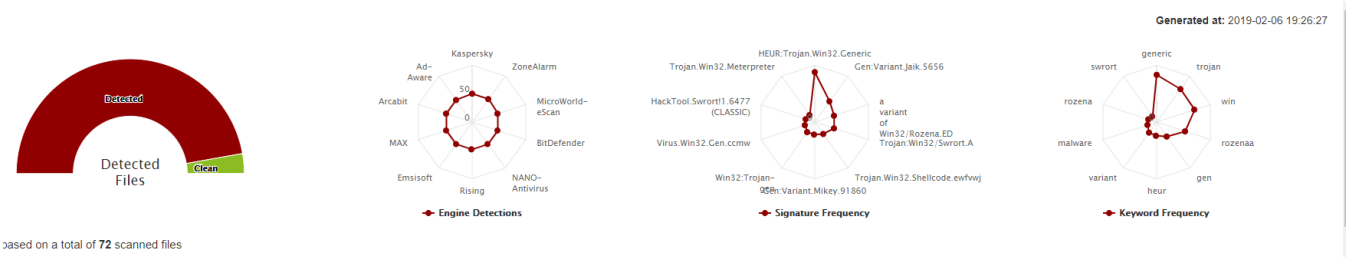


Figure 23. Charts in the VirusTotal report

2.5 Plugin: Injectotal

2.5.1 Execution

The Injectotal plugin calculates and generates a report with comparison charts and cumulative statistics from the virustotal analysis results. The arguments to execute the plugin are described below.

Table 13. Injectotal plugin arguments

Arguments	Type	Description
--injtotal-directory/-injtotaldir	String	A directory with virustotal results. **The plugin assumes that the files scanned from the virustotal plugin will be under the following directory structure <method>\<shellcode>\<filename> in order to be able to generate a meaningful.**

The execution of the injectotal plugin is pretty straight forward as the user has little interaction with the plugin execution.

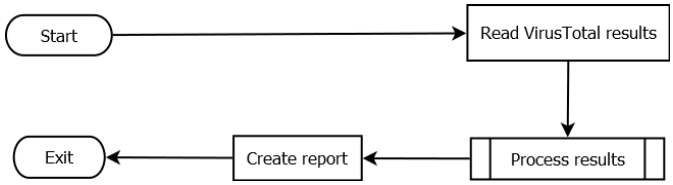


Figure 24. Flow chart of the execution of the Injectotal plugin

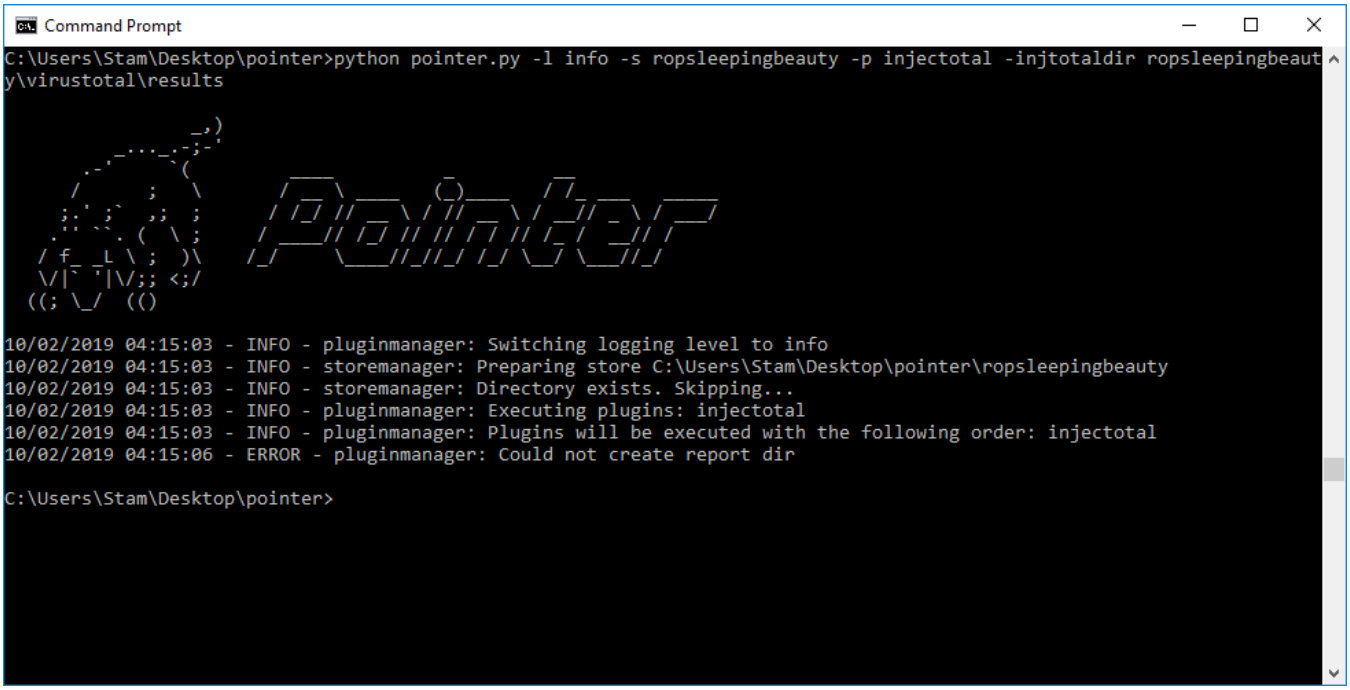


Figure 25. Execution of the Injectotal plugin in command line

2.5.2 Report

The plugin generates a report with a series of comparison charts. The report sections are described below.

Table 14. Chart titles and descriptions of charts in Injectotal report

Chart Titles	Description
Most effective antivirus engine	A series of pie charts displaying the top 10 antivirus engine with the most detections for each method / shellcode combination
Shellcode detection per antivirus engine	A series of bar charts comparing the number of detections of each antivirus per shellcode for each method
Method detection per antivirus engine	A series of bar charts comparing the number of detections of each antivirus per method for each shellcode
Evasion rate per shellcode	A series of bar charts comparing the evasion and unique evasion ratios of each shellcode and for each method for all files scanned
Evasion rate per method	A series of bar charts comparing the evasion and unique evasion ratios of each method and for each shellcode for all files scanned
Evasion rate per file	A series of bar charts comparing the evasion and unique evasion of each method per file and for each shellcode

Samples screenshots from the report are provided below.

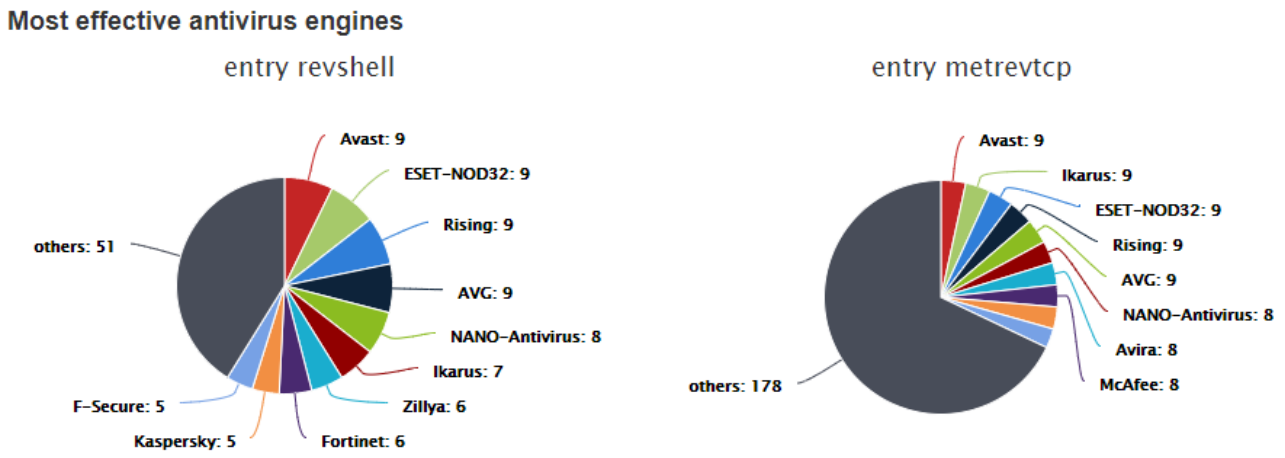


Figure 26. Most effective antivirus engines section in Injectotal report

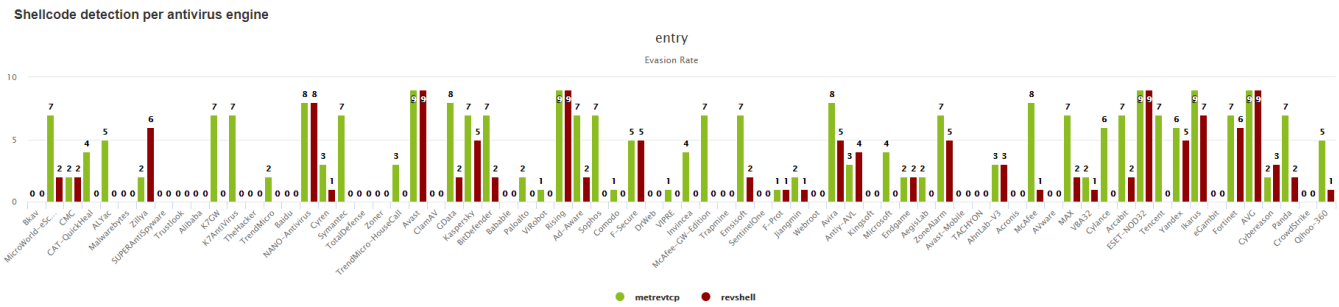


Figure 27. Shellcode detection per antivirus engine section in Injectotal report

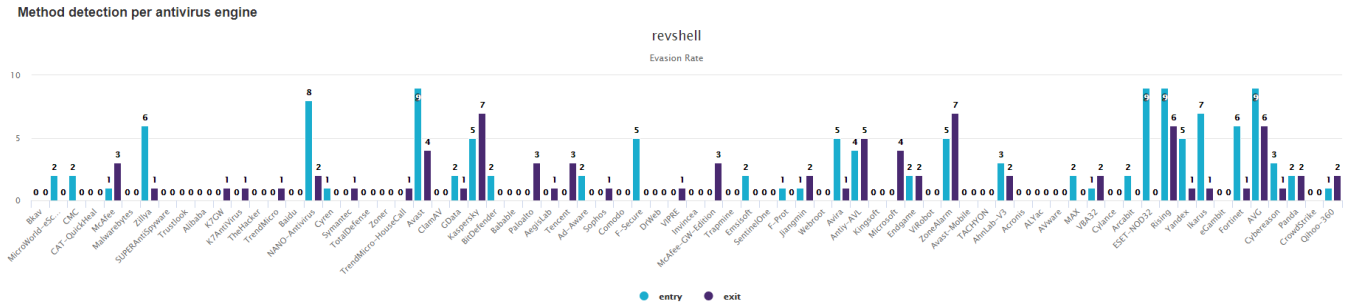


Figure 28. Method detection per antivirus engine section in Injectotal report

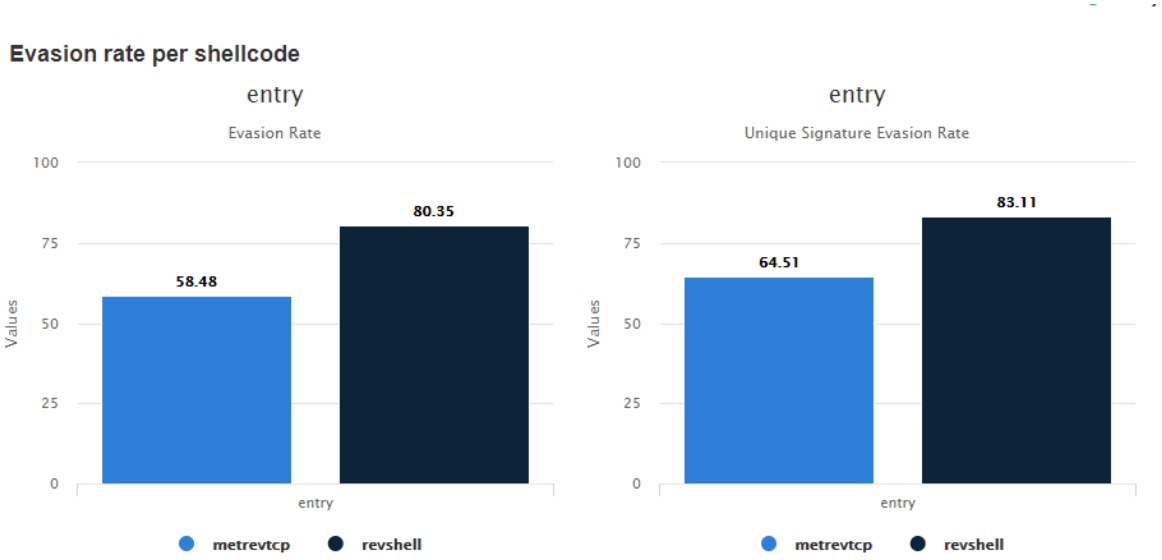


Figure 29. Evasion rate per shellcode section in Injectotal report

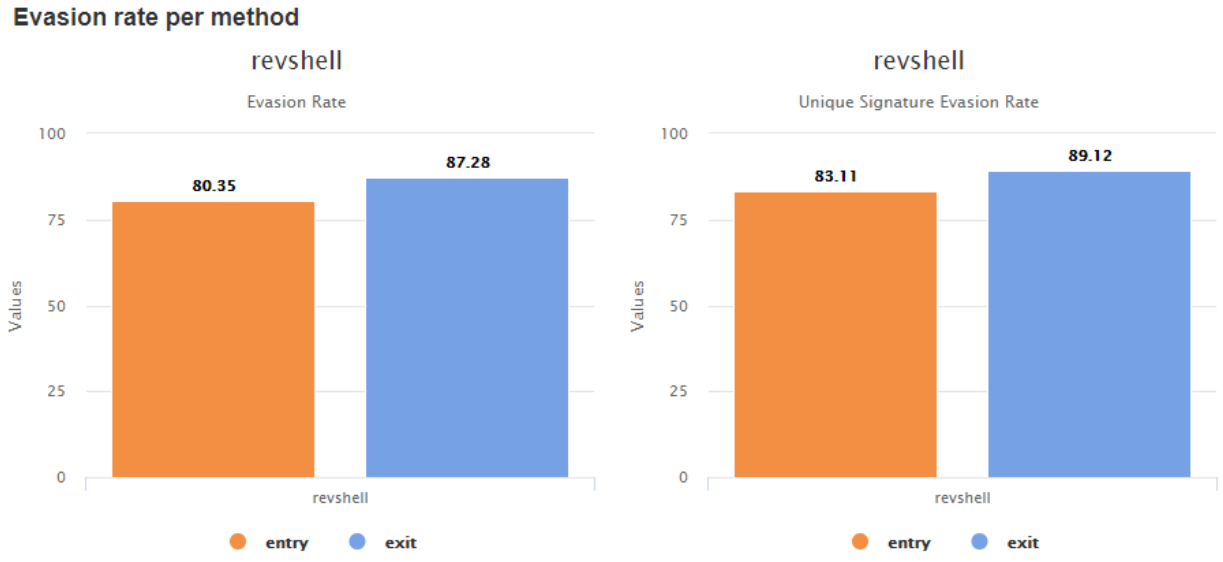


Figure 30. Evasion rate per method section in Injectotal report

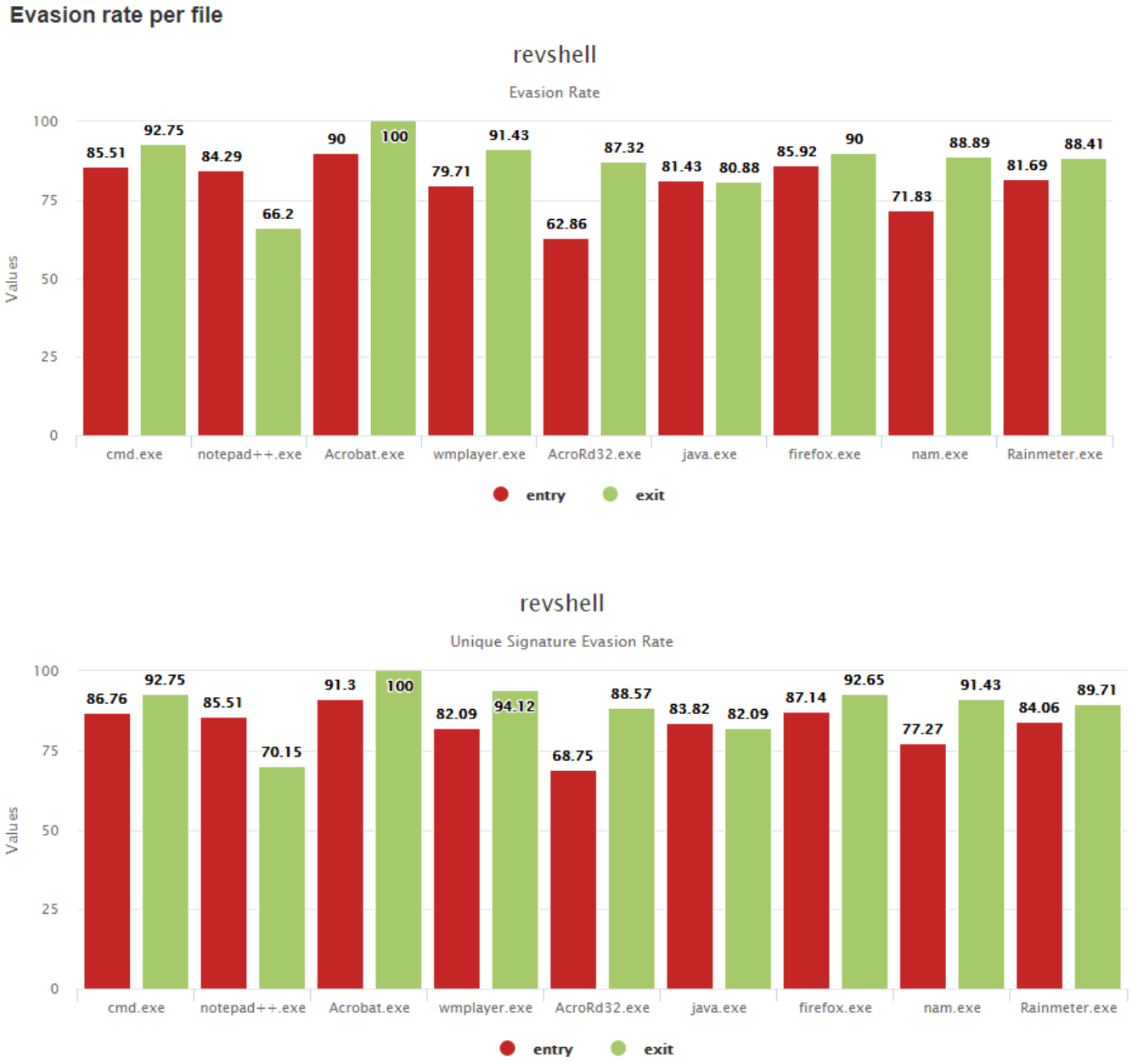


Figure 31. Evasion rate per file section in Injectotal report

2.6 Creating a new plugin

A template and example plugin are provided with this tool. However in this section we are going to note the minimum requirements that are needed in order to expand the tool functionality with new plugins.

```
import argparse
import logging

plugin_name= __name__.split(".")[-1]
plugin_type = 'PRESENTER'

def get_arguments(parser):
    group = parser.add_argument_group(plugin_name, 'Description of plugin here Type:%s',%PRESENTER)
    group.add_argument('--template_arg1',destination='template_arg1')
    group.add_argument('--template_arg2',destination='template_arg2')
    return parser

def arguments_check(plugin_args):
    if plugin_args['template_arg1'] is None:
        return "Argument template_arg1 must be provided"
    return None

def process(*args, **kwargs):
    logging.info("Name: %s" %plugin_name)
    logging.info("Type: %s" %plugin_type)
    logging.info("kwargs: %s"%kwargs)
    logging.info("kwargs: %s"%kwargs)
    logging.info(kwargs['args'])
    logging.info(kwargs['plugin_dir'])
```

Figure 32. Template plugin code snippet

The new plugin must have at least the following elements in order to be executed successfully by the tool:

Table 15. New plugin requirements

Requirement	Explanation
Import of argparse module	The argparse module must be imported as it is mandatory for a plugin to return a group of arguments even if that group is empty.
Import of logging module	Logging module must be imported and used within the plugin in order to ensure that the plugin provides sufficient information during execution.
Variable plugin_type	The variable plugin_type must be set to one of the following values: GENERATOR, ANALYZER, PRESENTER.
Function get_arguments	The function **get_arguments** must exist in order to provide the plugin arguments to the tool and allow the user to control the execution of the plugin.
Function arguments_check	The function **argument_check** must exist. The function is called prior to the plugin execution and is responsible for validating the user provided arguments.
Function process	The function **process** must exist. The function is responsible for executing the plugin functionality according to the user provided arguments.

If the plugin generates a report then the report must be placed under the directory ***<store_name>\<plugin_name>\report\<report name>.html*** otherwise it will not be identified by the tool. During each execution the tool looks for the report directory in each plugin directory and creates a link to every html file (plugin report) that exists inside them. In this way the tool cumulative report is updated with new information provided by the plugins as more plugins are executed.

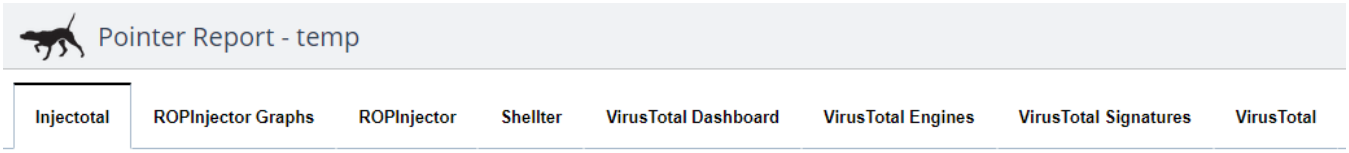


Figure 33. Report navigation panel dynamically generated at each run of the tool depending on the plugin reports that exist in the store

3 Use Cases

In order to evaluate and confirm the effectiveness of ROPInjector and Shellter we have selected 9 32-bit executables referenced in the next table for which we will run our tests.

Table 16. List of PE files used as carriers in the experiments

File Name	File Size (KB)	File Version	Hash (SHA256)
Acrobat.exe	650	19.10.20069.49826	067c6f0396600b725030db136f7db6d30d8706bd9e1f3a07cee4931ed2a02d91
AcroRd32.exe	1423	11.0.8.4	ed820c61c179fa27bb63305b5c18dbe913aea38cecc27835d3b3e51007e7d575
cmd.exe	305	6.3.9600.16384	e1a080e61fb1baf0da629d34baee6f0f9d0e0337bf6ced9f4b3ab9b1c23d91ba
firefox.exe	439	63.0.3.6892	76e344a43910a45679f208f1414bd720ca8efe5ca207d44179737da30aad090b
java.exe	187	8.0.192.12	b51c64c7ef4544dd04a76781e8be5b22482e7908b945528b08c9da73f07b4e4e
nam.exe	1828	1.0a11a	5d329bb39ba744cdba5e1afe107551c18ba0acd46cb6764391024a73aa2d583f
notepad++.exe	2783	7.6.0.0	c517690b5c9a83515b2d6aae6297990fc26ada6f06497507af714b0f0ea4ee96
Rainmeter.exe	39	2.4.0.1678	00c8f2b58ffb318cf1031f58f4fe86a73bcb9716c7072012114bd42f157dd071
wmplayer.exe	163	12.0.9600.19145	4e776d1969e18339bbc345ea281be3ebde034a4168e7266f247c1e004f544da8

Regarding the selected shellcode, we will be using the popular **reverse TCP shell** and **reverse TCP meterpreter** of the Metasploit Framework [4].

To put some perspective in the numbers that follow we are including screenshots from the VirusTotal Online Service for the aforementioned payloads when generated as a PE from the MSF.

The evasion rate of both payloads is **27.53 %** as seen from the screenshots below.

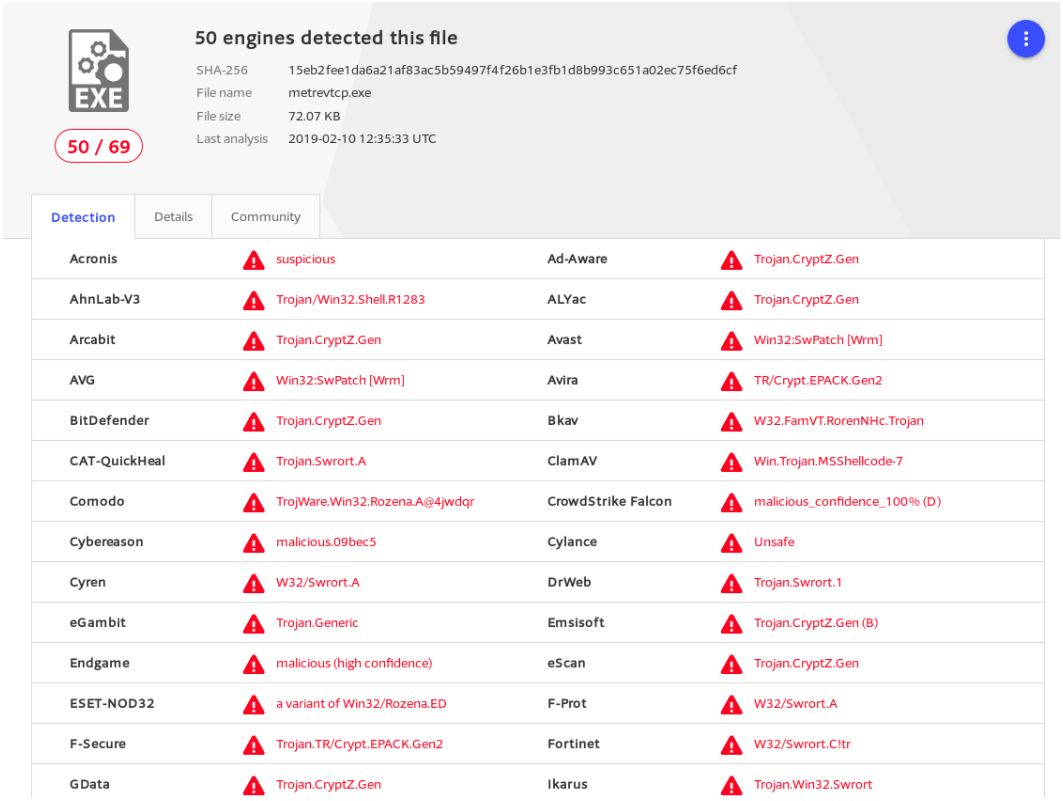


Figure 34 Detection rate of the Meterpreter Reverse TCP payload

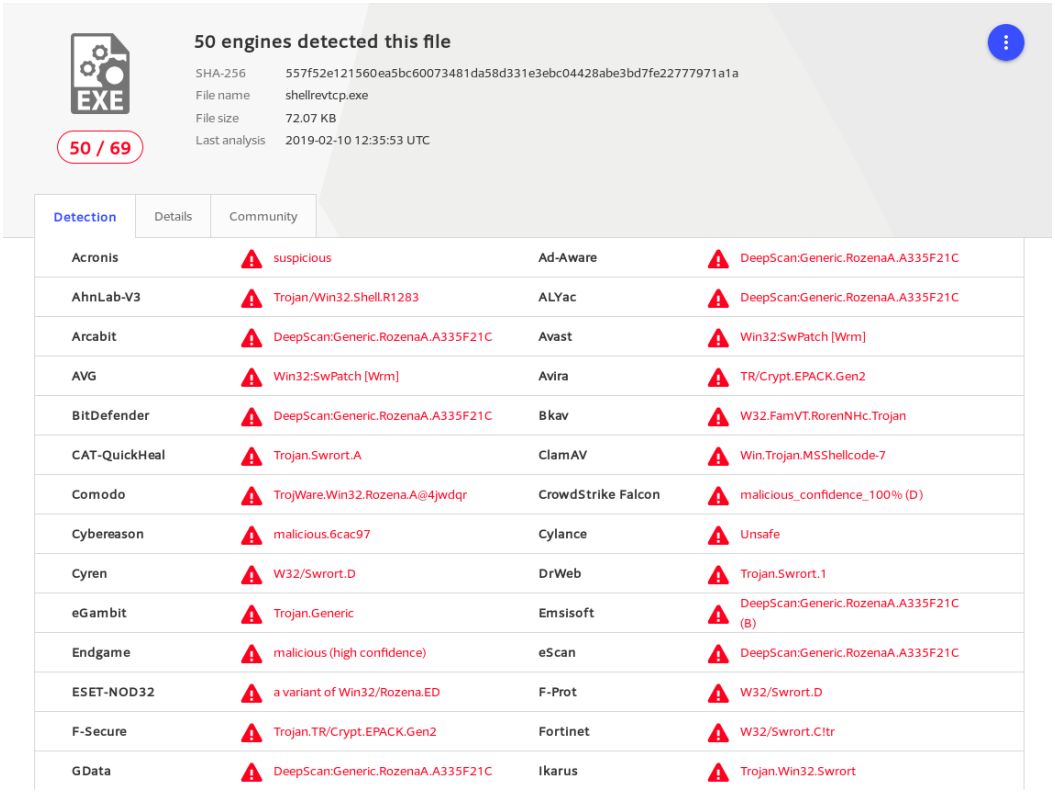


Figure 35. Detection rate of the Shell Reverse TCP payload

Graphs and information in the next two sections have all been generated automatically by the tool.

3.1 Use Case: ROPInjector

In this section we will demonstrate the usage of the tool by use-casing it with the ROPInjector shellcode injector. We analyze the various patch methods and techniques used by this shellcode injector and come with conclusions regarding its ability to evade AV solutions.

3.1.1 Primary Pathing Methods

For each of the aforementioned PE and each shellcode we have tested 4 patching methods listed in the following table, resulting in a total of 72 samples.

Table 17. List of ROPInjector patching methods tested

Patch Method	Description
ROP Entry	The executable file is patched with the shellcode unrolled, converted to ROP and the entry point before the original PE code.
ROP Exit	The executable file is patched with the shellcode unrolled, converted to ROP and the entry point before the original program's exit (hook ExitProcess or exit).
Shellcode Entry (norop nounroll)	The executable file is patched with the shellcode intact and the entry point before the original PE code.
Shellcode Exit (norop nounroll)	The executable file is patched with the shellcode intact and the entry point before the original program's exit (hook ExitProcess or exit).

Statistics and information regarding the patched PE are generated in each run of the ROPInjector. In the next table we have included the statistics for the sample files analyzed using the method ROP Entry and the shellcode reverse TCP shell. In the below results the reverse tcp shellcode consisted of **193** instructions out of which **139** were replaced with gadgets by the ROPInjector.

Table 18. Statistics from ROPInjector using the method ROP Entry and the shellcode reverse TCP shell for the carrier files

PE Name	PE Size (KB)	Candidate Gadgets Found in PE	Gadgets Injected	Gadgets Used From PE	Gadgets Injected (%)	Gadgets Used from PE Candidate Gadgets (%)
Acrobat.exe	650	338	78	12	86.67	3.55
AcroRd32.exe	1423	5599	67	26	72.04	0.46
cmd.exe	305	608	76	16	82.61	2.63
firefox.exe	439	1634	65	28	69.89	1.71
java.exe	187	1148	76	18	80.85	1.57
nam.exe	1828	3515	67	25	72.83	0.71
notepad++.exe	2783	7778	55	41	57.29	0.53
Rainmeter.exe	39	11	83	0	100.00	0.00
wmplayer.exe	163	60	82	3	96.47	5.00

In the next table we have included the statistics for the sample files analyzed using the method ROP Entry and the shellcode meterpreter reverse TCP. In the below results the shellcode consisted of **187** instructions out of which **129** were replaced with gadgets by the ROPInjector.

Table 19. Statistics from ROPInjector using the method ROP Entry and the shellcode meterpreter reverse TCP for the carrier files

PE Name	PE Size	Candidate Gadgets	Gadgets	Gadgets Used	Gadgets	Gadgets Used from PE

	(KB)	Found in PE	Injected	From PE	Injected (%)	Candidate Gadgets (%)
Acrobat.exe	650	338	77	8	90.59	2.37
AcroRd32.exe	1423	5599	67	19	77.91	0.34
cmd.exe	305	608	76	9	89.41	1.48
firefox.exe	439	1634	63	23	73.26	1.41
java.exe	187	1148	75	11	87.21	0.96
nam.exe	1828	3515	67	19	77.91	0.54
notepad++.exe	2783	7778	55	33	62.50	0.42
Rainmeter.exe	39	61	81	1	98.78	1.64
wmplayer.exe	163	60	80	2	97.56	3.33

We tested the evasion ratio of the infected files using the **virustotal** and **injectotal** plugins. The evasion results regarding each file method and shellcode combination can be seen on the following graphs.

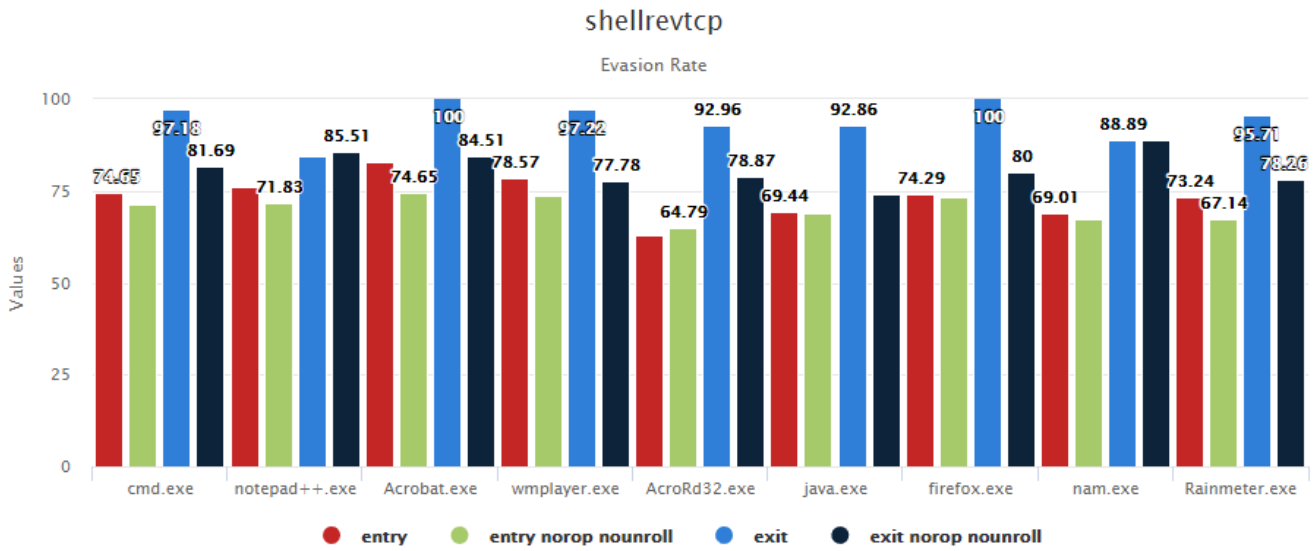


Figure 36. Comparison of evasion ratios of ROPInjector for the reverse shell payload per file and methods entry, entry norop nounroll, exit, exit norop nounroll

Table 20. Evasion ratios of ROPInjector for the reverse shell payload per file and methods entry, entry norop nounroll, exit, exit norop nounroll

File Name	entry	entry norop nounroll	exit	exit norop nounroll
cmd.exe	74.65	71.43	97.18	81.69
notepad++.exe	76.06	71.83	84.51	85.51
Acrobat.exe	82.86	74.65	100	84.51
wmplayer.exe	78.57	73.61	97.22	77.78
AcroRd32.exe	62.86	64.79	92.96	78.87
java.exe	69.44	69.01	92.86	74.29
firefox.exe	74.29	73.24	100	80
nam.exe	69.01	67.14	88.89	88.73
Rainmeter.exe	73.24	67.14	95.71	78.26

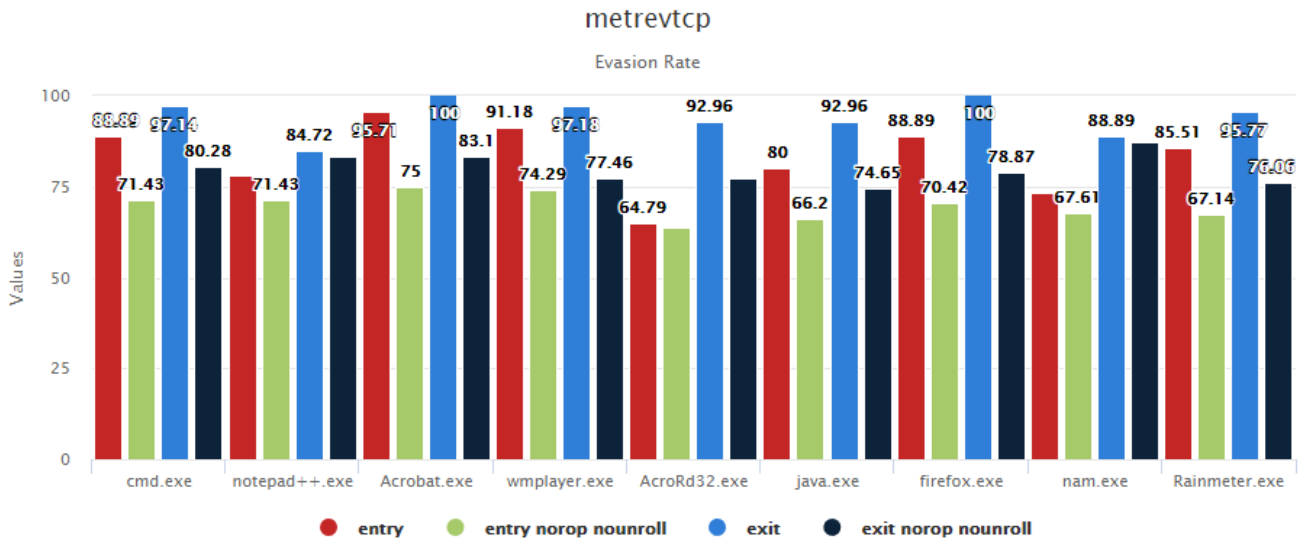


Figure 37. Comparison of evasion ratios of ROPInjector for the meterpreter payload per file and methods entry, entry norop nounroll, exit, exit norop nounroll

Table 21. Evasion ratios of ROPInjector for the meterpreter payload per file and methods entry, entry norop nounroll, exit, exit norop nounroll

File Name	entry	entry norop nounroll	exit	exit norop nounroll
cmd.exe	88.89	71.43	97.14	80.28
notepad++.exe	77.94	71.43	84.72	83.1
Acrobat.exe	95.71	75	100	83.1
wmplayer.exe	91.18	74.29	97.18	77.46
AcroRd32.exe	64.79	63.89	92.96	77.14
java.exe	80	66.2	92.96	74.65
firefox.exe	88.89	70.42	100	78.87
nam.exe	73.13	67.61	88.89	87.32
Rainmeter.exe	85.51	67.14	95.77	76.06

We can observe that the executables generated by the ROPInjector using the ROP Exit method achieve the highest evasion ratio. To put that in perspective cumulative evasion statistics have are calculated for each of the shellcode method combinations from the injectotal plugin. In the analyzed sample the ROP Exit method scored an impressive **94.37** and **94.38** evasion ratio for the reverse tcp and reverse meterpreter payloads. The percentage shoots even higher to **96.02** and **96.03** if consider our defined metric of unique evasion ratio. It is also important to note that the unique evasion ratio for the rest of the patching methods increases their evasion by **6-8%**. As a reminder a unique evasion ratio assumes that detections performed with the same signature name from different antivirus engines are considered as one single detection from one antivirus engine as different products may use the same signature, heuristics or behavior database.

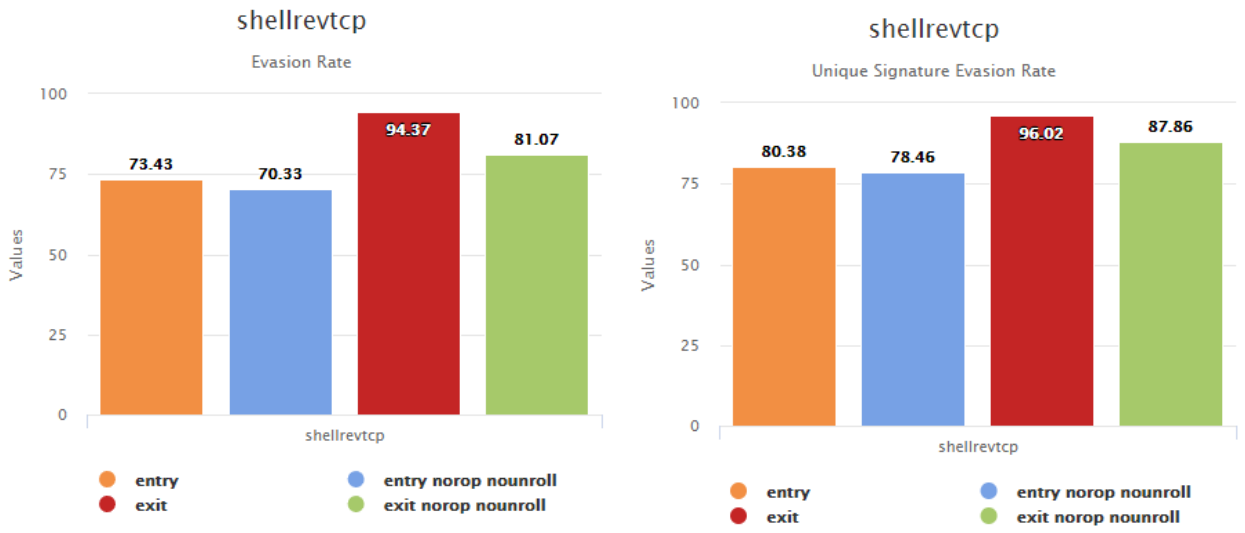


Figure 38. Evasion and unique evasion ratio of ROPInjector for the reverse shell payload per method.

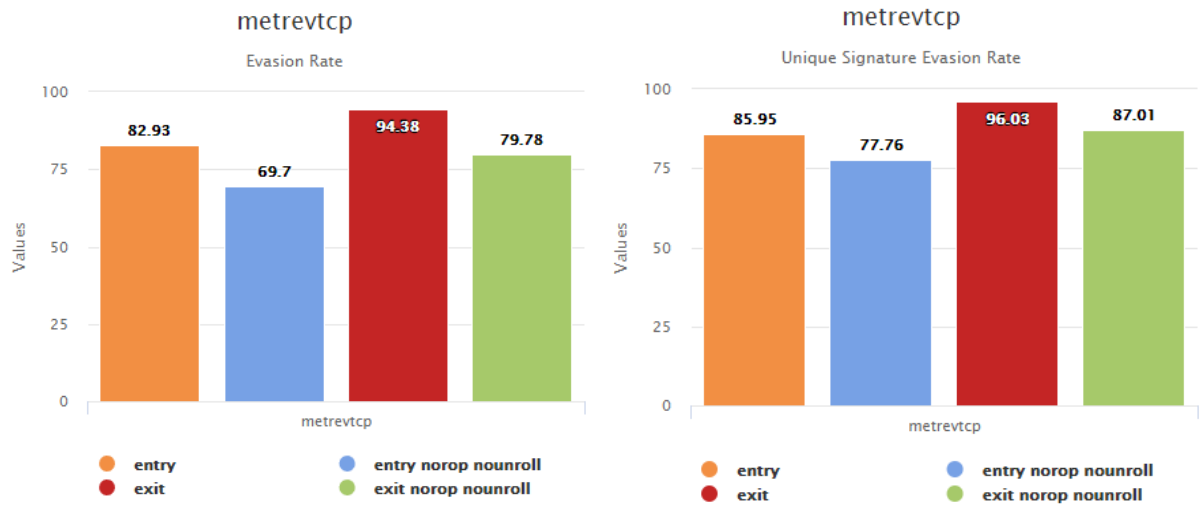


Figure 39. Evasion and unique evasion ratio of ROPInjector for the reverse meterpreter payload per method

Statistics regarding the effectiveness of each antivirus engine against the ROPInjector have also been calculated by the injectotal plugin. For the analyzed sample and the methods ROP Entry and ROP Exit we include the top 10 most effective antivirus engines per payload used. The numeric values next to the engine name refer to the number of files that were detected from the solution for each method and shellcode combination. The detections of the remaining engines are summarized in the other category. It is also important to note that for each case a maximum of 9 detections is possible (as for each case 9 files are generated by the ropinjector).

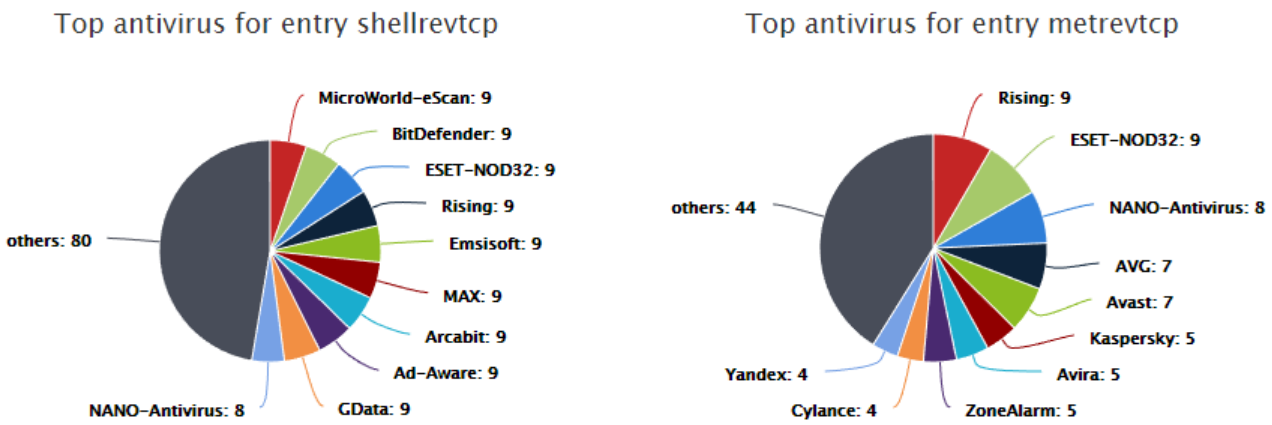


Figure 40. Most effective antivirus engines for the ROP Entry method

Table 22. Antivirus detections for the ROP Entry method

Shell Reverse TCP payload		Metepreter Reverse TCP Payload	
Engine	**Files Detected**	**Engine**	**Files Detected**
MicroWorld-eScan	9	Rising	9
BitDefender	9	ESET-NOD32	9
ESET-NOD32	9	NANO-Antivirus	8
Rising	9	AVG	7
Emsisoft	9	Avast	7
MAX	9	Kaspersky	5
Arcabit	9	Avira	5
Ad-Aware	9	ZoneAlarm	5
GData	9	Cylance	4
NANO-Antivirus	8	Yandex	4
ALYac	8	MicroWorld-eScan	3
Cybereason	8	BitDefender	3
F-Secure	7	Ad-Aware	3
Avira	6	Emsisoft	3
AVG	6	Arcabit	3
Avast	6	MAX	3
Kaspersky	5	GData	3
ZoneAlarm	5	Cybereason	3
Ikarus	3	CMC	2
Antiy-AVL	3	Zillya	2
Yandex	3	Jiangmin	2
Qihoo-360	3	Antiy-AVL	2
CMC	2	Endgame	2
VIPRE	2	AhnLab-V3	2
Endgame	2	Ikarus	2
Zillya	2	Qihoo-360	2

AhnLab-V3	2	**McAfee**	1
McAfee	1	**Cyren**	1
K7GW	1	**F-Prot**	1
K7AntiVirus	1	**VBA32**	1
F-Prot	1	**Bkav**	0
Cyren	1	**CAT-QuickHeal**	0
Jiangmin	1	**VIPRE**	0
VBA32	1	**TheHacker**	0
Bkav	0	**K7GW**	0
TotalDefense	0	**K7AntiVirus**	0
CAT-QuickHeal	0	**Invincea**	0
Cylance	0	**Baidu**	0
TheHacker	0	**Babable**	0
TrendMicro	0	**Symantec**	0
Baidu	0	**TotalDefense**	0
Babable	0	**TrendMicro-HouseCall**	0
Symantec	0	**Paloalto**	0
TrendMicro-HouseCall	0	**ClamAV**	0
Paloalto	0	**Alibaba**	0
ClamAV	0	**ViRobot**	0
Alibaba	0	**AegisLab**	0
ViRobot	0	**Trustlook**	0
SUPERAntiSpyware	0	**Sophos**	0
Trustlook	0	**Comodo**	0
Comodo	0	**F-Secure**	0
DrWeb	0	**DrWeb**	0
Invincea	0	**TrendMicro**	0
McAfee-GW-Edition	0	**McAfee-GW-Edition**	0
Fortinet	0	**Trapmine**	0
Trapmine	0	**SentinelOne**	0
Sophos	0	**Webroot**	0
Webroot	0	**Fortinet**	0
Kingsoft	0	**Kingsoft**	0
AegisLab	0	**SUPERAntiSpyware**	0
Avast-Mobile	0	**Avast-Mobile**	0
Microsoft	0	**Microsoft**	0
Acronis	0	**TACHYON**	0
AVware	0	**Acronis**	0
TACHYON	0	**ALYac**	0
Malwarebytes	0	**AVware**	0
Panda	0	**Malwarebytes**	0
Zoner	0	**Panda**	0

Tencent	0	**Zoner**	0
SentinelOne	0	**Tencent**	0
eGambit	0	**eGambit**	0
CrowdStrike	0	**CrowdStrike**	0

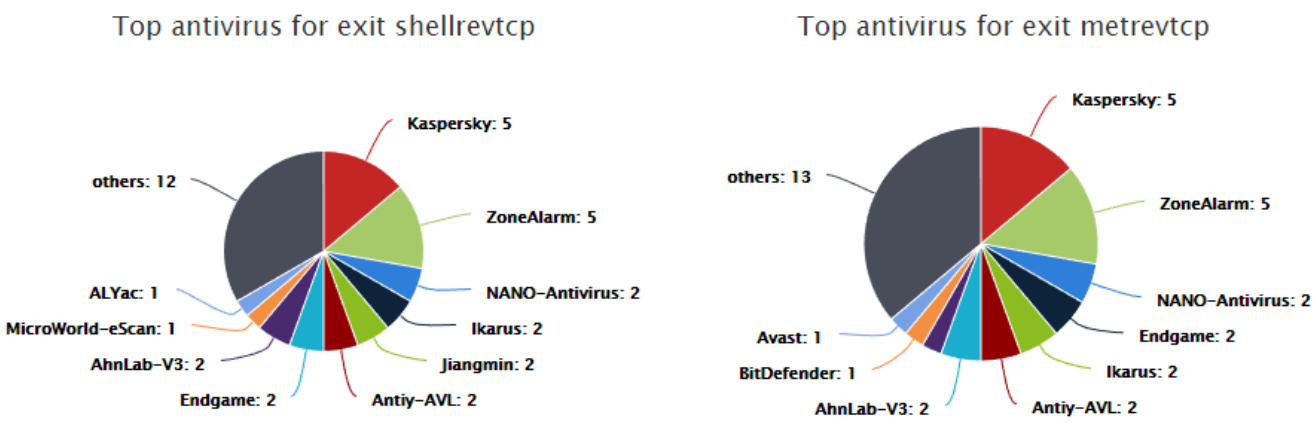


Figure 41. Most effective antivirus engines for the ROP Exit method

Table 23. Antivirus detections for the ROP Exit method

Shell Reverse TCP payload		Metepreter Reverse TCP Payload	
Engine	**Files Detected**	**Engine**	**Files Detected**
Kaspersky	5	**Kaspersky**	5
ZoneAlarm	5	**ZoneAlarm**	5
NANO-Antivirus	2	**NANO-Antivirus**	2
Ikarus	2	**Endgame**	2
Jiangmin	2	**Ikarus**	2
Antiy-AVL	2	**Antiy-AVL**	2
Endgame	2	**AhnLab-V3**	2
AhnLab-V3	2	**MicroWorld-eScan**	1
MicroWorld-eScan	1	**BitDefender**	1
ALYac	1	**Avast**	1
BitDefender	1	**Rising**	1
Avast	1	**Emsisoft**	1
Ad-Aware	1	**F-Secure**	1
Emsisoft	1	**Jiangmin**	1
F-Secure	1	**ALYac**	1
Avira	1	**Avira**	1
MAX	1	**Arcabit**	1
Arcabit	1	**MAX**	1
Rising	1	**Ad-Aware**	1
Yandex	1	**Yandex**	1

GData	1	**GData**	1
AVG	1	**AVG**	1
Bkav	0	**Cybereason**	1
CMC	0	**Bkav**	0
CAT-QuickHeal	0	**CMC**	0
Malwarebytes	0	**CAT-QuickHeal**	0
VIPRE	0	**McAfee**	0
AegisLab	0	**Cylance**	0
Trustlook	0	**VIPRE**	0
K7GW	0	**Trustlook**	0
K7AntiVirus	0	**K7GW**	0
TrendMicro	0	**K7AntiVirus**	0
Baidu	0	**Invincea**	0
Babable	0	**Baidu**	0
F-Prot	0	**Babable**	0
Symantec	0	**Cyren**	0
TotalDefense	0	**Symantec**	0
TrendMicro-HouseCall	0	**TotalDefense**	0
ClamAV	0	**TrendMicro-HouseCall**	0
Alibaba	0	**ClamAV**	0
ViRobot	0	**Alibaba**	0
Tencent	0	**ViRobot**	0
Comodo	0	**AegisLab**	0
DrWeb	0	**Comodo**	0
Zillya	0	**DrWeb**	0
Invincea	0	**Zillya**	0
McAfee-GW-Edition	0	**TrendMicro**	0
Fortinet	0	**McAfee-GW-Edition**	0
Trapmine	0	**Trapmine**	0
TheHacker	0	**TheHacker**	0
Cyren	0	**F-Prot**	0
Webroot	0	**Webroot**	0
Kingsoft	0	**Fortinet**	0
SUPERAntiSpyware	0	**Kingsoft**	0
Avast-Mobile	0	**SUPERAntiSpyware**	0
Microsoft	0	**Avast-Mobile**	0
Sophos	0	**Microsoft**	0
Acronis	0	**TACHYON**	0
McAfee	0	**Sophos**	0
AVware	0	**Acronis**	0
TACHYON	0	**VBA32**	0
VBA32	0	**AVware**	0

Cylance	0	**Malwarebytes**	0
Panda	0	**Panda**	0
Zoner	0	**Zoner**	0
ESET-NOD32	0	**ESET-NOD32**	0
SentinelOne	0	**Tencent**	0
eGambit	0	**SentinelOne**	0
Cybereason	0	**eGambit**	0
Paloalto	0	**Paloalto**	0
CrowdStrike	0	**CrowdStrike**	0
Qihoo-360	0	**Qihoo-360**	0

An interesting and scary observation on the above number is that many of the popular commercial solutions used widely in organizations fail to score high in the detection of the infected binaries while less popular solutions are successfully in detecting them.

By looking at the virustotal report for both the ROP Entry and the ROP Exit we can identify some interesting results. In the case of ROP Entry among the most frequent keywords identified is the rozena – a malware that uses a meterpreter payload to communicate to its CnC. However for the same files this is not the case for the ROP Exit method. Specifically the keyword rozena appears 68 times in the ROP Entry method and only 2 in the ROP Exit. This is an indication that the ROP Exit method not only achieves better evasion results but also succeeds in better hiding the nature of the payload it executes.

Figure 42. Word cloud of signature keywords for the ROP Entry (left) and ROP Exit (right) methods

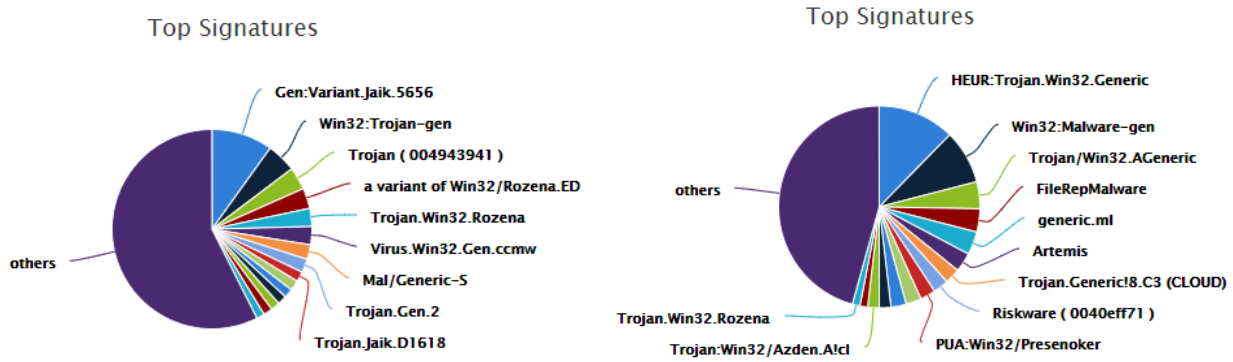


Figure 43. Pie charts of top signatures for the ROP Entry (left) and ROP Exit (right) methods

A noticeable difference also exists in the number of engines that have been triggered in each of the two methods. Specifically a total of **50** engines (**69.4%**) have been triggered for at least one file for the ROP Entry method while only **34** (**47.2%**) have been triggered for the ROP Exit method.

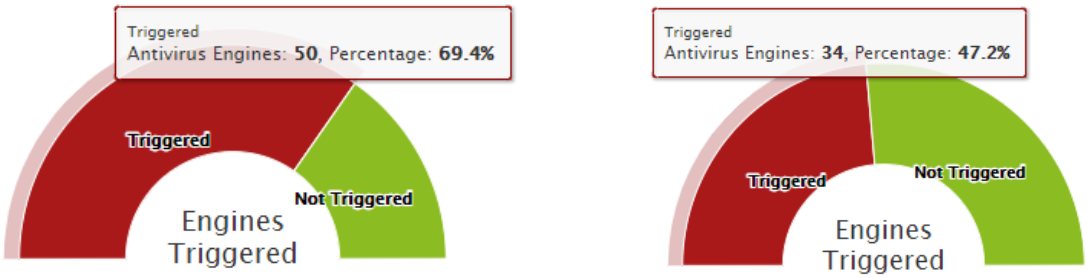


Figure 44. Semi pie charts of Engines triggered for the ROP Entry (left) and ROP Exit (right) methods

3.1.2 Padded Shellcode

Based on the evasion results of each method we tested above, we observed that by delaying the execution of the shellcode as is the case in the ROP Exit method, we have achieved very good evasion results. In order to confirm this idea we have designed the following experiment. We have appended at the beginning of both payloads a padding of random assembly instructions, namely: inc, dec, and, or, xor, not, cmp, neg, sub, add. We repeated this process for 50, 100 and 250 instructions and created 3 variations for each of the aforementioned payloads. We tested again the evasion results of the ROP Entry patching method for the 6 shellcodes by generating a total of 54 samples.

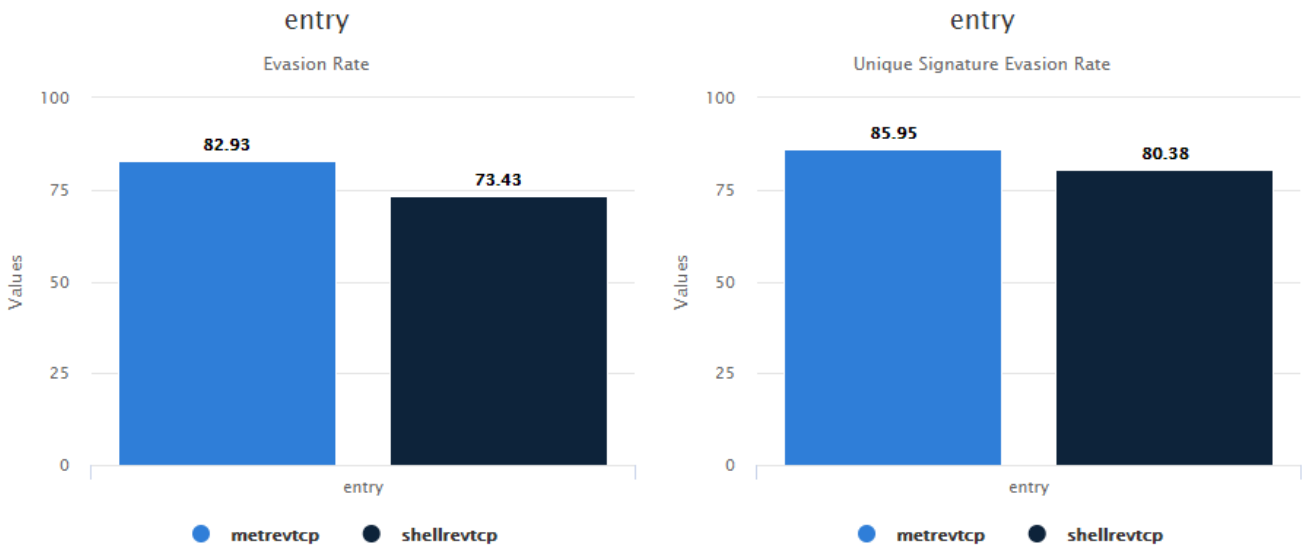


Figure 45. Evasion and unique evasion ratio of ROPInjector for the ROP Entry patching method and padded shellcode

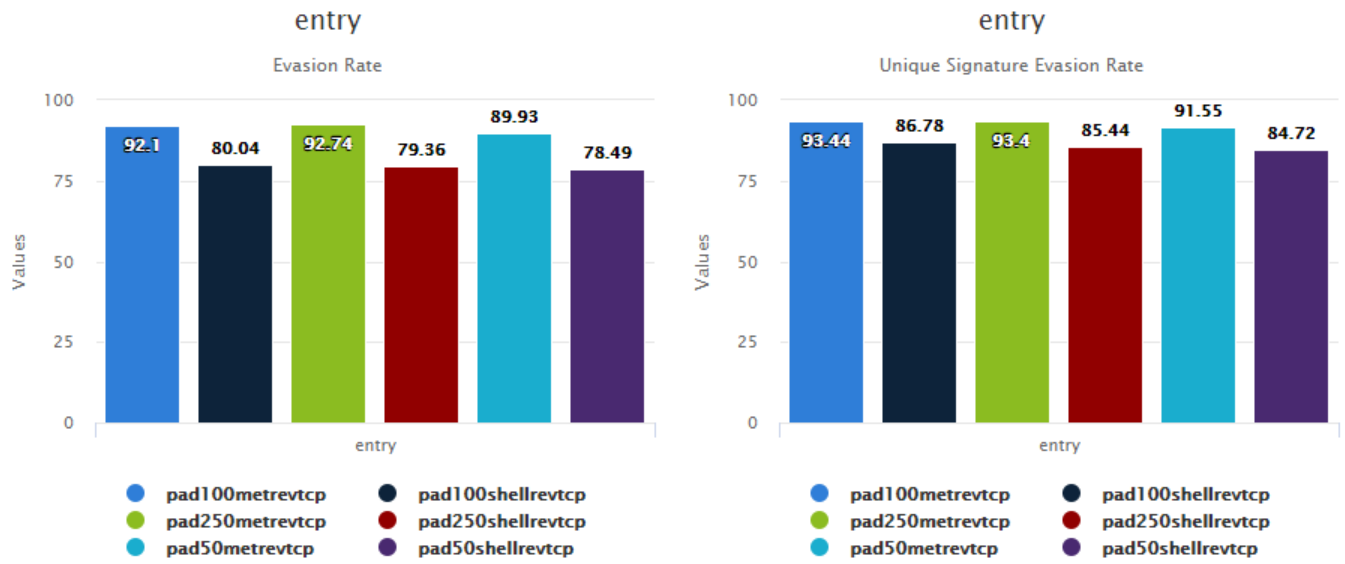


Figure 46. Evasion and unique evasion ratio of ROPInjector for the ROP Entry patch method and padded shellcode

The padded shellcode proved to improve the evasion ratio of the ROP Entry method by approximately **7-10%**. Overall by padding the shellcode we have managed to achieve evasion results close to the ones of the ROP Exit method. Especially for the case of the meterpreter reverse TCP payload the evasion has increased to a **92.74%** for 250 instructions appended before the shellcode execution.

3.1.3 Sleep before payload execution

We also we make use of the sleep capability of the ROPInjector that delays the execution of the shellcode. We test this for 5, 60 and 300 seconds delay and the methods ROP Entry, ROP Exit, Entry norop nounroll and Exit norop nounroll resulting in a total 216 samples. The delay introduced before the execution is passed to the injected payload has no effect in the evasion rates of the files. It is possible that this is due to the fact that some antivirus engines bypass sleep times when analyzing code.

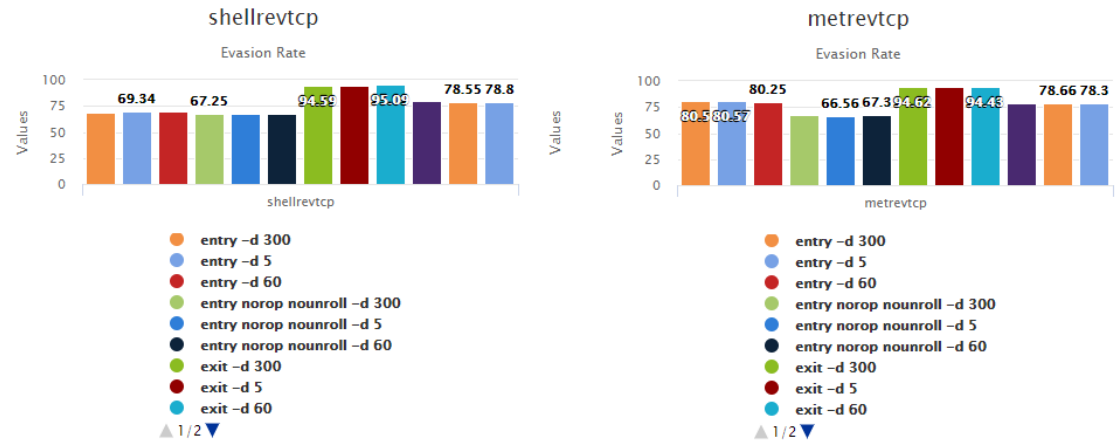


Figure 47. Evasion and unique evasion ratio of ROPInjector with delay introduced before the execution of the shellcode

3.1.4 Hiding vs Deleting the Certificate

For the last experiment with this tool we are going to test if there is any difference between hiding and deleting the certificate from a signed PE. For this reason the results that follow are only tested the 5 signed PEs from our original sample namely: Acrobat.exe, AcroRd32.exe, firefox.exe, java.exe, notepad++.exe.

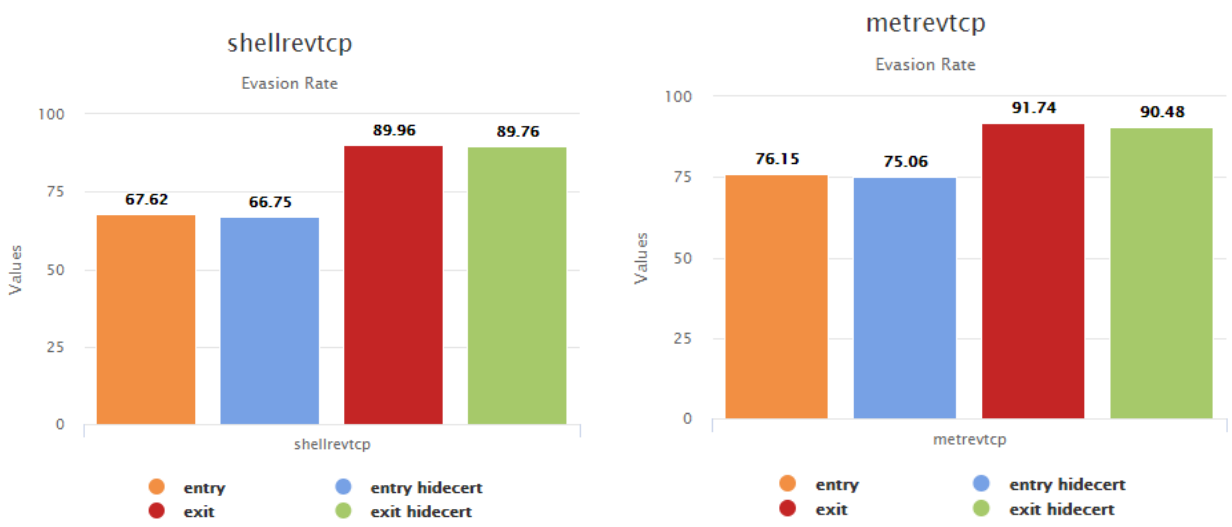


Figure 48. Evasion ratio of ROPInjector for the shellcodes shellrevtcp and metrevtcp using a hide and delete patching method

Although the evasion rates for these methods are very close, in the case were the certificate is deleted from the infected files the evasion rate is slightly but steadily above the evasion rates of the files that have the certificate hidden.

3.1.5 Conclusions

From our experiments above we have deduced the following interesting conclusions:

- The ROP Exit method is the most effective one comparing to the rest. The reason for that might be that the antivirus engines mostly analyze the instructions during the entry of executables.
- Popular commercial Antivirus Solutions used widely in major organizations fail in many cases to detect the infected file.
- The nature of the payload is hidden, as the signature used to detect it are in the majority of the cases either too generic or inaccurate.
- A basic shellcode obfuscation helps to increase the evasion results. In our experiments just calling a number of instructions before the execution of the malicious payload helped increasing the evasion results 7-10 % for the ROP Entry method.
- A call to sleep is ineffective in helping increase the evasion ratio regardless of the time parameter.
- The certificate of signed binaries should be deleted from the infected file as hiding it will leave overlay data which are detectable using static analysis.

3.2 Use case: Shellter

In this section we analyze the effectiveness of Shellter shellcode injector.

3.2.1 Stealth vs No Stealth

We are going to test the two primary patching methods used by Shellter and identify the evasion ratios in each case. The patching methods are described in the next table.

Table 24. List of Shellter patching methods tested

Patch Method	Description
Stealth	If stealth mode is enabled then Shellter preserves original functionality of the infected PE file.
No Stealth	If stealth mode is disabled then Shellter does not preserve the original functionality of the infected PE.

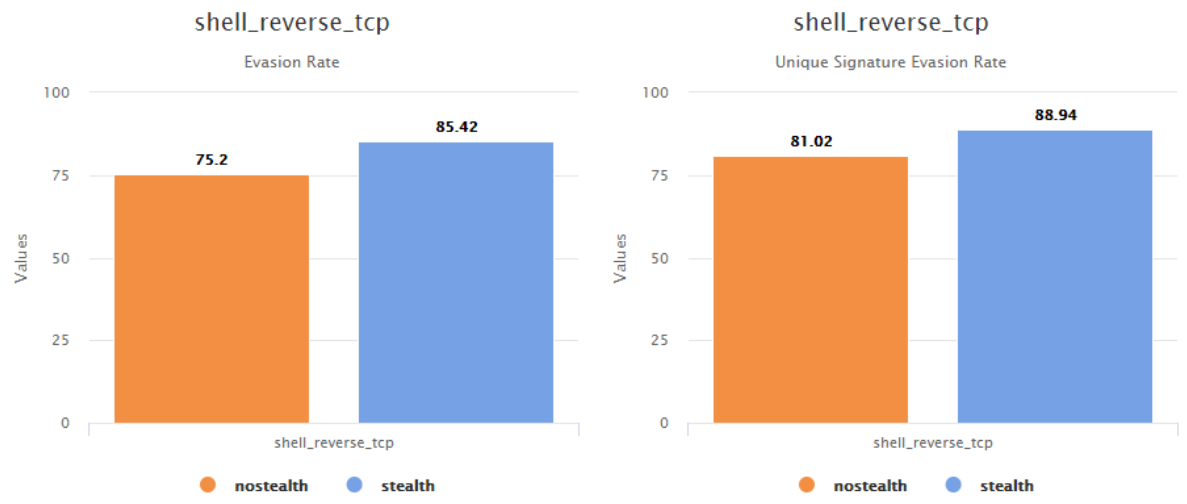


Figure 49. Evasion ratio and unique evasion ratio of stealth and no stealth mode for payload shell_reverse_tcp

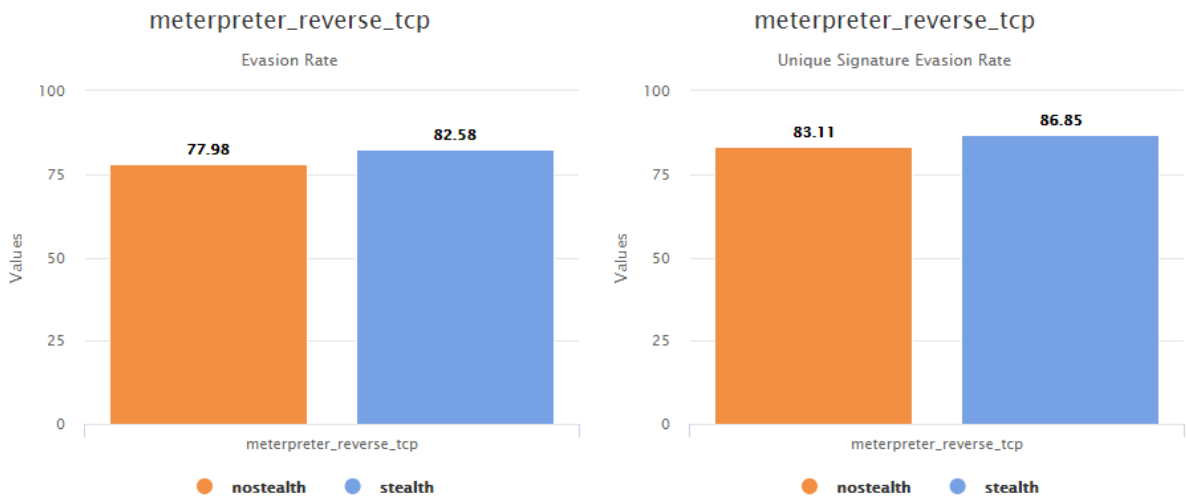


Figure 50. Evasion ratio and unique evasion ratio of stealth and no stealth mode for payload meterpreter_reverse_tcp

It is obvious by the charts above that returning to the normal execution flow after the payload execution achieves very good anti-detection results. In the analyzed sample using stealth mode Shellter scored **85.42** and **82.58** for the shell_reverse_tcp and the meterpreter_reverse_tcp payloads respectively and **88.94** and **86.85** if we consider the metric unique evasion ration. Another important thing to note is that shellter is not very effective when it comes to hiding the payload nature or the method used. Among the top triggered signatures and signature keywords are the following.

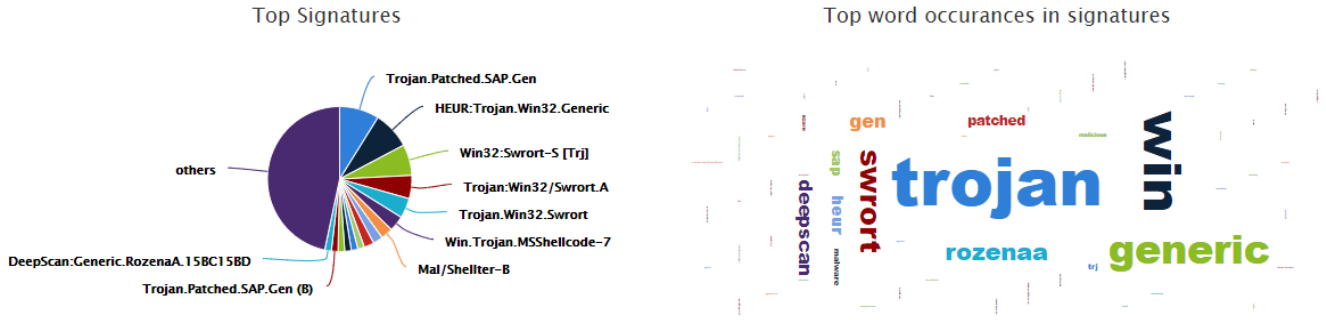


Figure 51. Top signatures and signature keywords for stealth and no stealth methods

From the charts above can see that the antivirus engines are able to detect that the files analyzed are patched, that the file is possibly a form of shellcode and there are even signatures specifically for shellter.

3.2.2 Polymorphic Junk Code

Another run using stealth and no stealth mode was performed but this time the –junk argument has been used. The –junk flag enables polymorphic junk code and produces a more complex output. This type of code added also serves for timing-out some emulators and sandboxes provided that you might have to wait for a few seconds before the payload gets executed. Evasion results used from this method are provided below.

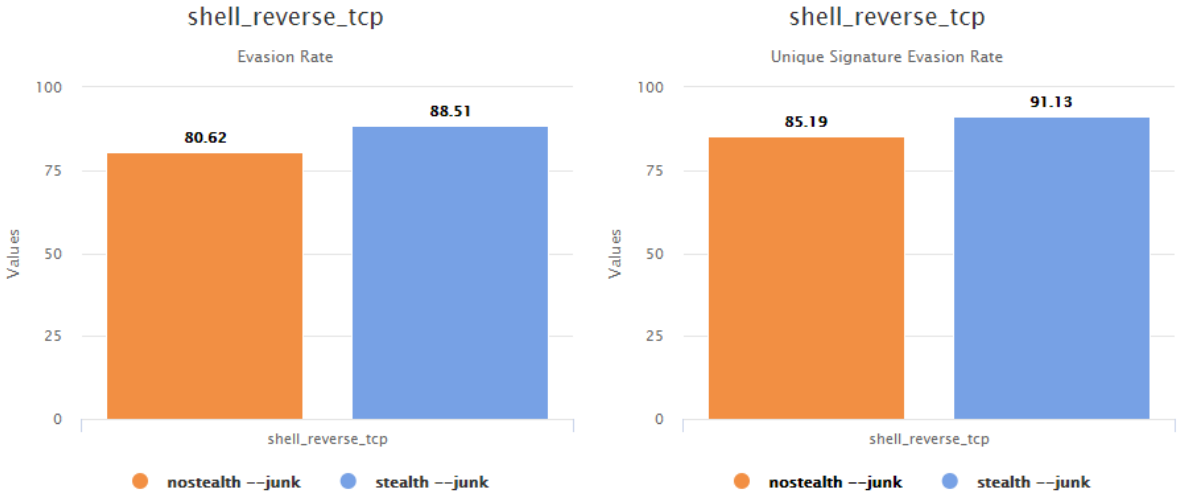


Figure 52. Evasion ratio and unique evasion ratio of stealth and no stealth mode using junk before payload execution for payload shell_reverse_tcp

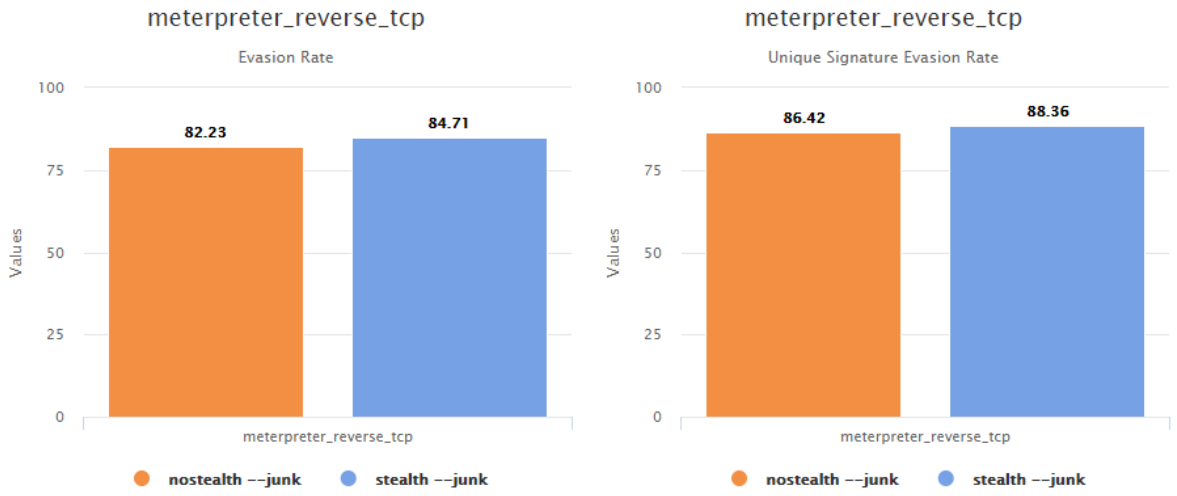


Figure 53. Evasion ratio and unique evasion ratio of stealth and no stealth mode using junk before payload execution for payload meterpreter_reverse_tcp

It is clear from the results above that junk method increases the evasion ratio by a difference of approximately **5% in** each of the payloads and each of the methods. Similar to the previous experiment the analyzed samples are detected as patched and related with shellcode however this time there is no signature related to the detection of the tool used like before.

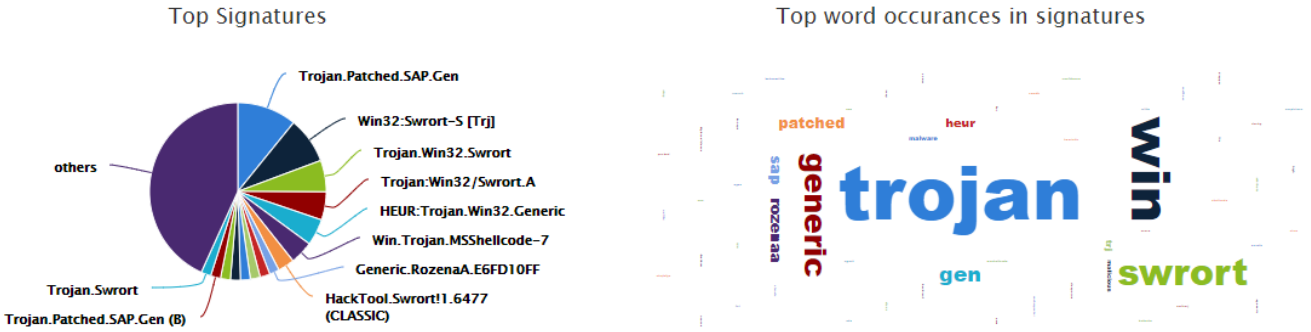


Figure 54. Top signatures and signature keywords for stealth and no stealth methods with the junk flag enabled

3.2.3 Encoded Payloads

We have also tested Shellter's capability to apply an encoding layer to the payload. Specifically Shellter provides the following four encoding functions - **XOR, AND, NOT, SUB**. We are testing these encoding functions in order to identify whether one of them is superior to the other and whether their usage helps increase the evasion ratio of the analyzed carrier files.

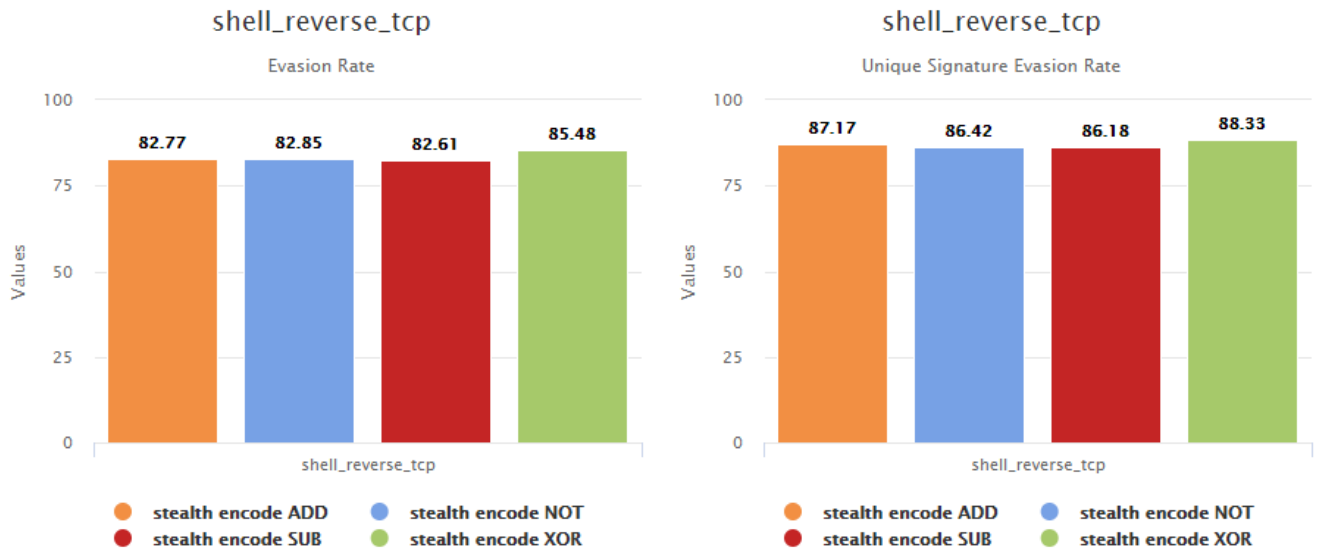


Figure 55. Evasion ratio and unique evasion ratio of XOR, AND, NOT, SUB operations for payload shell_reverse_tcp

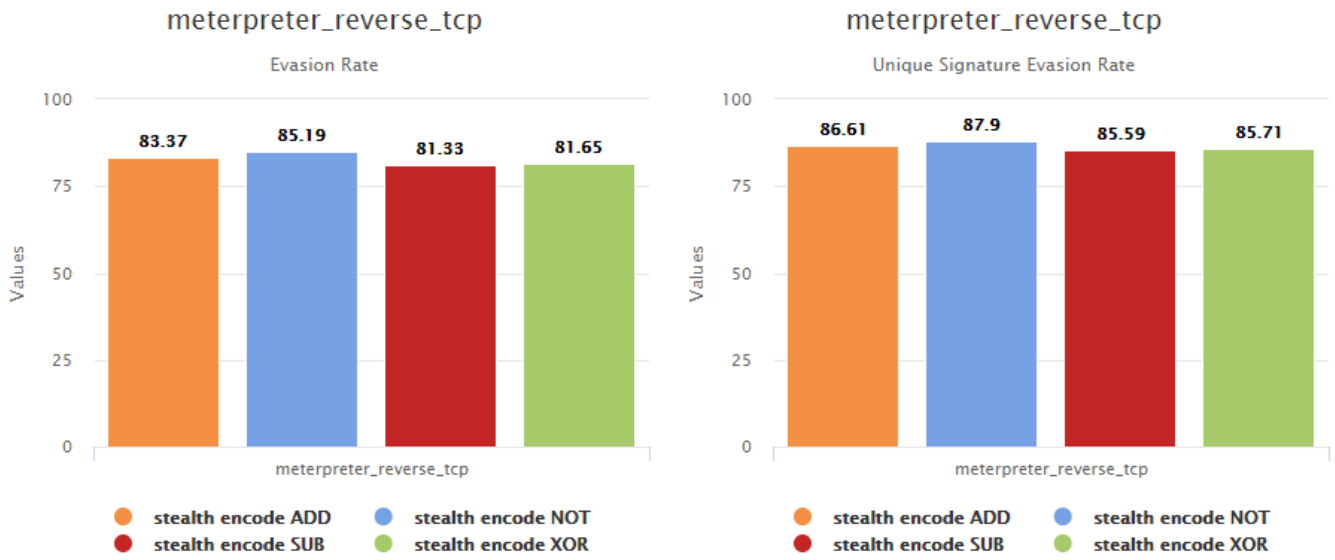


Figure 56. Evasion ratio and unique evasion ratio of XOR, AND, NOT, SUB operations for payload meterpreter_reverse_tcp

The encoding function evasion ratios are similar to one another with the difference between them being less than approximately **3%**. It would also seem from the results above that there is no superior encoding function as for different payloads different encoding functions score higher in the evasion ratio scale. It is also surprising that the encoding functions are actually reducing the evasion ratio of the infected binaries.

We are going to repeat the process but this time we are going to combine multiple iterations of the encoding functions. Results are again mixed with the evasion being lower than before.

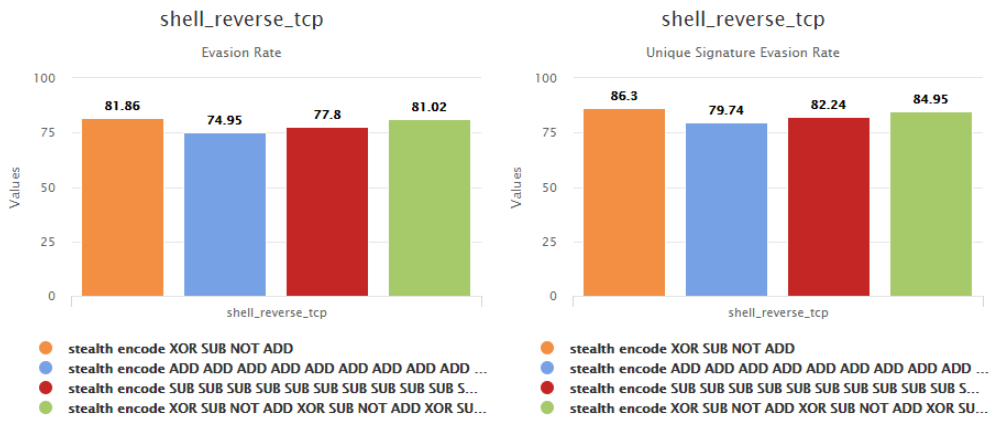


Figure 57. Evasion ratio and unique evasion ratio of combined encoding operations for payload shell_reverse_tcp

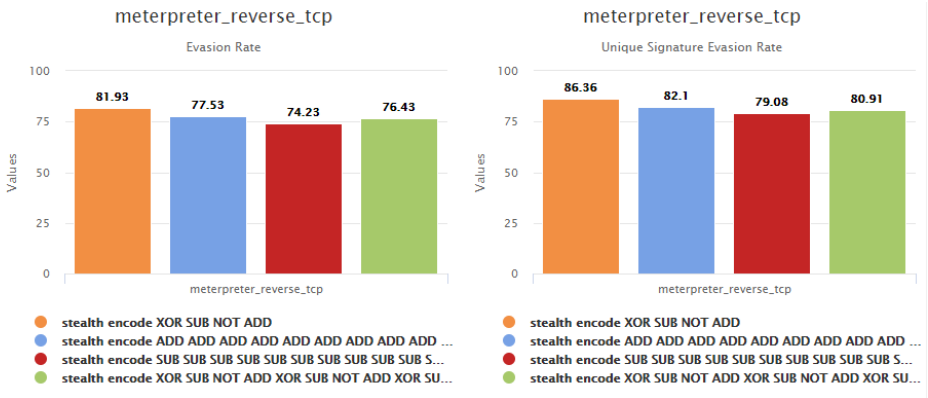


Figure 58. Evasion ratio and unique evasion ratio of combined encoding operations for payload meterpreter_reverse_tcp

In an attempt to understand this behavior we identified that file evasion ratios have great differences. For example for the **meterpreter_reverse_tcp** payload the evasion of **cmd.exe** is **94.12%** while the evasion of **notepad++** is **71.43%**.

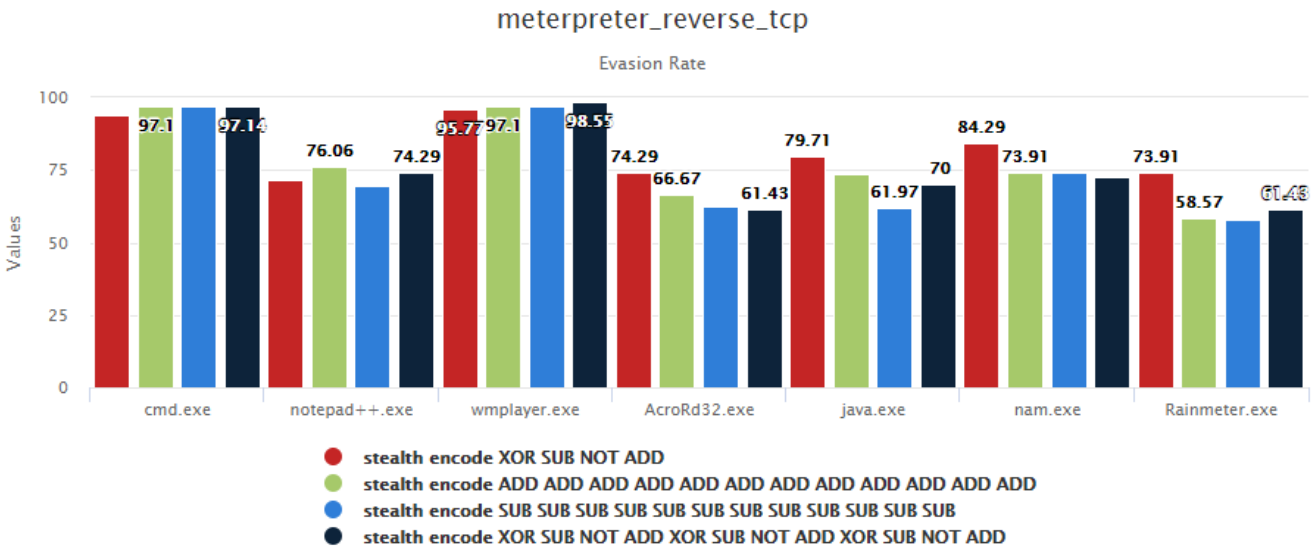


Figure 59. Evasion ratio and unique evasion ratio of combined encoding operations per file for payload meterpreter_reverse_tcp

Also by the signature analysis we can observe that when using any type of encoding more of the carrier files are being detected as patched and the shellter related signatures have higher trigger rate.

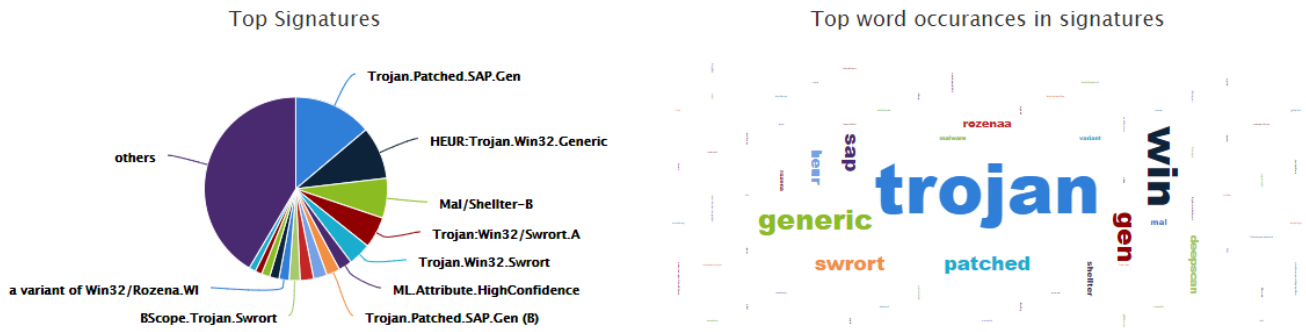


Figure 60. Top signatures and signature keywords for encoding methods XOR,ADD,NOT, SUB

3.2.4 Combination of methods

Finally we combine the most evasive method from each category in order to attempt to reach the limits of the Shellter shellcode injector and create a sample of very evasive binaries. Therefore we will be using the methods stealth with junk before payload execution and XOR payload encoding. The results can be seen below.

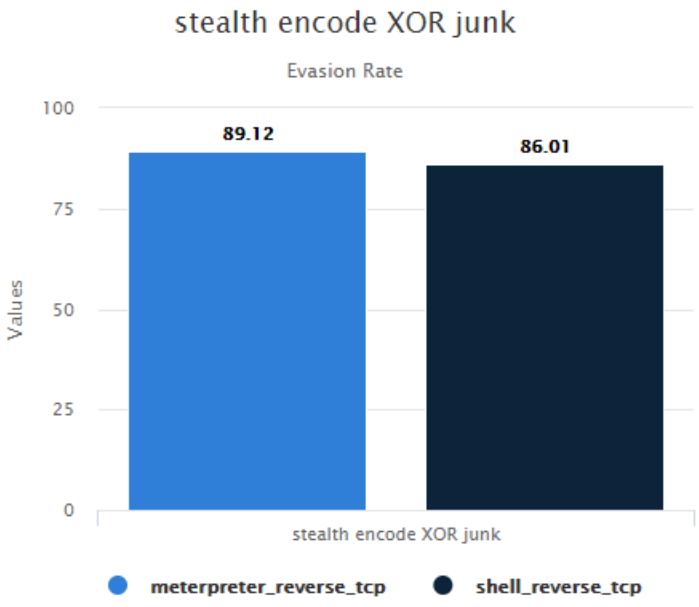


Figure 61. Evasion ratios of combined methods stealth, XOR and junk for payload shell_reverse_tcp and meterpreter_reverse_tcp

This combination indeed achieves the highest evasion ratio so far when compared to the previous tests performed.

3.2.5 Conclusions

From our experiments with Shellter we have deduced the following interesting conclusions:

- Shellter seems to achieve the highest evasion ratio when executed with a combination of methods used. The highest evasion ratio was 89.12% for the payload meterpreter_reverse_tcp.
- Different carrier files may achieve very diverse evasion ratios.
- In several analyzed samples the infected file was detected as patched and in some occasions shellter was even successfully detected by the antivirus engine as a signature indicating that the tool behavior has been analyzed and modeled.
- Payload encoding does not help increase the evasion ratio of infected files when used on each own. In fact the evasion ratio was reduced and more signatures were detected as patched.

A cumulative table comparing the results from the previous sections is included below.

Table 25. Evasion ratios of Shellter methods

	Evasion of shell_reverse_tcp (%)	Evasion of meterpreter_reverse_tcp (%)
Stealth	85.42	82.58
No Stealth	75.2	77.98
Stealth Junk	88.51	84.71
No Stealth Junk	80.62	82.23
Stealth Encode ADD	82.77	83.37
Stealth Encode NOT	82.85	85.19
Stealth Encode SUB	82.61	81.33
Stealth Encode XOR	85.48	81.65
Stealth Encode XOR SUB NOT ADD	81.86	81.93
Stealth Encode ADDx12	74.95	77.53
Stealth Encode SUBx12	77.8	74.23
Stealth Encode (XOR SUB NOT ADD)x3	81.02	76.43
Stealth Encode XOR Junk	86.01	89.12

4 Conclusions

Most antivirus engines rely on string signatures and heuristic analysis for the detection of malicious code. In the case of the shellcode injectors that have been examined in this Thesis, string signatures proved to be ineffective as a combination of polymorphic techniques and different patching methods would practically randomize the carrier files. Heuristic signatures have had more success in detecting infected files but behavioral profiling was easily bypassed when methods for emulating a benign behavior were utilized (junk or random code before the payload execution, return to normal execution flow after payload execution, execute payload during exit etc…).

Another noticeable fact is that many of the popular and effective solutions [5] according to public ranking have failed to detect the carrier files in several scenarios let alone detect accurately the nature of the payload. This is a worrying fact for organizations that still rely solely to antivirus solutions for their endpoint's protection.

From the analyzed shellcode injectors it seems that although being less popular and known, ROPInjector is more effective in evading detection than shellter. This is probably due to the fact that the execution of the malicious payload is broken down to ROP gadgets and injected in smaller parts in the carrier file. ROPInjector also attempts to use as much of the carrier's file code as possible and in case it is not able to, inject code in such a way as to emulate a real function. It should be noted that its evasion ratio three years following its release has only dropped 1-2% from the numbers presented in Blackhat 2015. [6]

Shellter on the other hand, although less effective in antivirus evasion offers a number of options for manipulating and transforming the payload before the injection on the carrier file. That said it is still a valid option for bypassing most of the antivirus engines but should be used with caution as a lot of the times the evasion results between samples were very diverse.

For the conclusions and comments regarding the effectiveness of antivirus engines and shellcode injectors it should also be taken into consideration that test were performed using two of the most popular shellcodes from MSF, a well-known and widely used exploitation framework. Using custom payloads would most likely, but without having evidence to support this, result in greater evasion ratios from both shellcode injectors. Further manipulating the payload before the injection is also bound to decrease detection as already proven in multiple scenarios. Summing up, the two key points for good evasion results seems to be a) the injection entry point (at which point in the carrier files execution is the malicious code executed) and b) the payload transformation (transforming or altering the payload to make it look as benign as possible).

Finally, the tool developed for the purpose of analyzing shellcode injector has still room from improvement. No work has been done for statically analyzing the carrier files. Injection and manipulation of a PE file is a very tricky process and a lot of the times the injection can easily be identified with static analysis methods. A potentially promising idea would be to develop a plugin that would perform static analysis checks (checksum calculation and comparison, check for overlay data, file metadata, checks with yara rules etc..).

References

[1] Giorgos Poulios. (2019, Jan.) Github. [Online]. **https://github.com/gpoulios/ROPInjector**

[2] kyREcon. (2019, Jan.) Shellter. [Online]. **https://www.shellterproject.com/**

[3] VirusTotal. (2019, Jan.) [Online]. **https://www.virustotal.com/#/home/upload**

[4] Metasploit. (2019, Jan.) [Online]. **https://www.metasploit.com/**

[5] Gartner. (2019, Jan.) Magic Quadrant Report. [Online]. **https://www.crowdstrike.com/resources/reports/2018-gartner-magic-quadrant-endpoint-protection-platforms/**

[6] George Poulios. (2019, Jan.) [Online]. **https://www.blackhat.com/docs/us-15/materials/us-15-Xenakis-ROPInjector-Using-Return-Oriented-Programming-For-Polymorphism-And-Antivirus-Evasion.pdf**

[7] Python. (1, 2019) [Online]. **https://www.python.org/**