RoAMer: Robust Automated Malware Unpacker

Fraunhofer
FKIE
Thorsten Jenke
Daniel Plohmann
Elmar Padilla

2019-04-03

Outline

Presentation agenda

Motivation

- In-depth analysis is a core building block for understanding malware
- Most of today's malware is packed or obfuscated
- Unpacking is a necessary first step for analysis
- Automation of unpacking is highly desirable

Previous Approaches

- There have been numerous previous approaches
- They are either...
  - not well tested against real-world malware
  - lacking in generality
  - lacking in evasion resilience
  - lacking in through-put
  - not released for the community

RoAMer

- Introducing the Robust Automated Malware Unpacker (RoAMer)
- It is a new automated generic unpacker
- We evaluated our approach with two diverse data sets
  - Malpedia
  - Malshare 2017

Development

- Basically my Master Thesis
- Supervised by Daniel Plohmann and later by Elmar Padilla

Presentation agenda

Unpacked vs. Dumped

- Unpacked
  - As close as possible to the original payload prior to packing
  - Typically achieved by intercepting execution after unwrapping
  - Can be executed as-is
- Dumped
  - Extracting system's memory segments containing the payload (also known as dumping)
  - In most cases cannot be run as-is
  - Only an approximation of the original malware
  - Initialized data fragments

(Dis-)Advantages

- Dumping is technically easier to achieve
- Static analysis often does not require a perfect reconstruction of the original
- Unpacked samples are easier to utilize in dynamic analysis
- Dumps contain run-time data, such as dynamic imports and decrypted strings
- There are approaches for reconstructing samples from memory dumps into an executable form
- Goal of RoAMer: enable static analysis by dumping malware

Types of Packers

- 6 Types of packers defined by Ugarte-Pedrero [3]
    - Type I: Simplest packer
    - Type II: Multiple simple packers in a line
    - Type III: Multiple simple packers in a tree
    - Type IV: Payload triggers packers
    - Type V: Payload code is mangled with packer code
    - Type VI: Decrypt on demand and then encrypted again

Presentation agenda

Idea

- Heavily inspired by the methodology of real-world analysts:
  1. Execute malware in sandbox environment
  2. Focusing on suspicious behavior like startup of new processes, sudden changes in memory sizes. . .
  3. Dump new and suspicious regions
  4. Decide whether to continue investigation or commence static analysis on dumps

Idea

- Assumption: New memory regions have to be allocated for malware to run
- This makes the payload directly observable and therefore "dumpable"

Filters

- Therefore, algorithm returns a set of dumps of suspicious regions
- Desired target dump among libraries, heap-sections, etc.
- Filters have to be set in place to find the desired dump

Limitations

- Userland-only
- Requires one point in time where the whole image is exposed in memory
- Only packers I through V comply to this
- Typical problems with native execution of malware (sleep, specific time, etc.)

Presentation agenda

System Design

- Written in Python 2.7 for Windows 7 32/64bit
- One component residing on the host system and the other in client
- Host part controls VM and interface to the user
- Client part (agent) is responsible for unpacking
- Interaction and observation with the memory is done through the Windows API

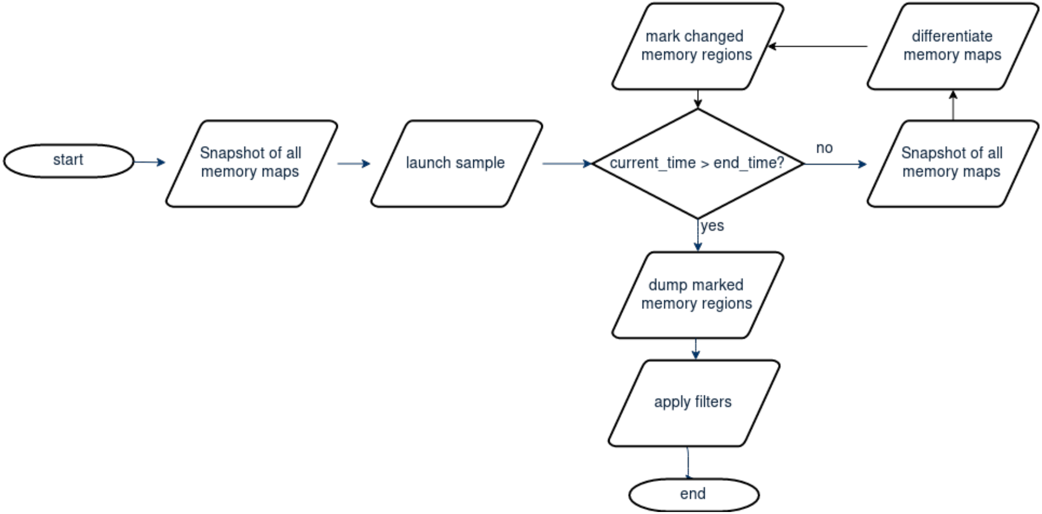
Output Filters

1. PE-header whitelist filter
2. Eliminate non-executable regions that are not adjacent to an executable region

Adressing Evasion Techniques

- Debugger detection
- Fingerprinting VM
- User Interaction
- Evading all other techniques: NtTerminateProcess-Hook

Workflow



start
Snapshot of all memory maps
launch sample
current_time > end_time?
no
Snapshot of all memory maps
differentiate memory maps
mark changed memory regions
yes
dump marked memory regions
apply filters
end

Presentation agenda

Metrices

- Correctness
- Precision
- Speed

Correctness

- Is the desired dump among the output?
- Number of additional dumps does not factor in
- True, False, No-changes

Precision

- How many undesired dumps are among the output?
- The additional amount of undesired dumps may become too high
- The higher the precision the higher the quality of the output

Speed

- The time that the methodology needs to unpack the malware
- Proposed method involves unsupervised execution of malware for predefined time
- Unfeasibility of algorithm grows with the amount of time needed

Speed

- Therefore speed is the amount of time passed until the first correct output dump is observable

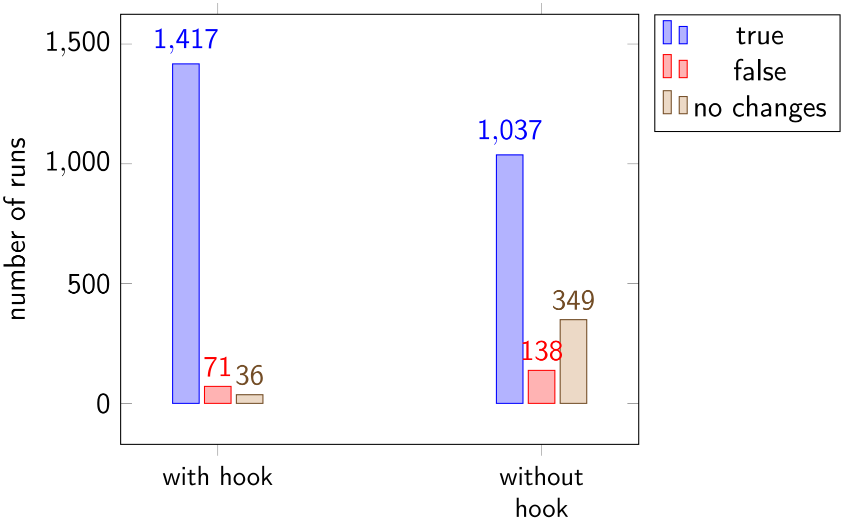
Setup

- Two datasets: Malpedia [2] and Malshare [1] 2017
- Malpedia
  - Well curated malware corpus
  - Ground truth through manually unpacked/dumped reference samples
- Malshare 2017
  - Every PE-file uploaded to Malshare in 2017
  - Contains also potentially goodware and not packed samples
  - No ground truth available
- Each sample runs for 10 minutes with and without hook

Malpedia Correctness

- TLSH to compare manually and automatically dumped samples
- Yara signatures from Malpedia's database to determine correctness

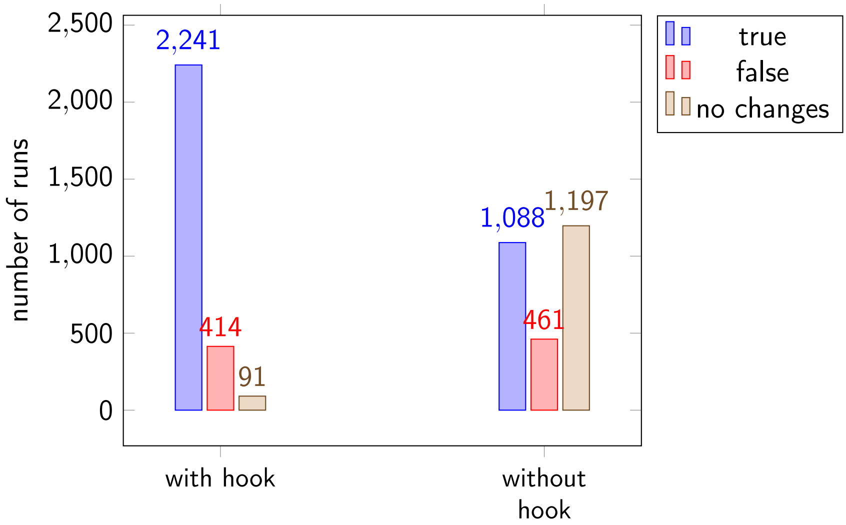
Malpedia Correctness



Malshare Correctness

- No ground truth available
- Comparison of original header vs. dumped header
- Therefore not packed samples and goodware considered incorrect

Malshare Correctness



Malpedia Precision

	Malpedia w/o Hook	Malpedia w/ Hook
min	1	1
25%	1	4
Average	29.7	19.1
Median	2	6
75%	5	12
max	17703	6313

Table: Number of dumps for Malpedia

Malpedia Precision

	Malpedia w/o Hook	Malpedia w/ Hook
min	0%	0%
25%	0%	0%
Average	217%	439%
Median	1%	29%
75%	33%	112%
max	41338%	91575%

Table: Overhead size for Malpedia

Malshare Precision

	Malshare w/o Hook	Malshare w/ Hook
min	1	1
25%	3	4
Average	7.8	11.29
Median	5	7
75%	7	13
max	89	171

Table: Number of dumps for Malshare

Malshare Precision

	Malshare w/o Hook	Malshare w/ Hook
min	0%	0%
25%	1%	3%
Average	200%	230%
Median	5%	25%
75%	48%	75%
max	112250%	112230%

Table: Overhead size for Malshare

Malpedia Speed

	Malpedia w/o Hook	Malpedia w/ Hook
Min	11	11
25%	11	11
Average	21.59	17.81
Median	11	11
75%	11	11
Max	641	626

Table: Dump timing for Malshare in seconds

Malshare Speed

	Malshare w/o Hook	Malshare w/ Hook
Min	11	11
25%	11	11
Average	17.02	15.39
Median	11	11
75%	16	11
Max	596	563

Table: Dump timing for Malshare in seconds

Discussion

- Hook increases correctness and speed
- Hook decreases precision
- Tradeoff between correctness, speed, and precision
- Enable or disable the hook according to circumstances

Presentation agenda

Conclusion

- RoAMer utilizes a technique commonly used by malware analysts
- Evaluated RoAMer against two data-sets
- With promising results
- Good basis for future work

Future Work

- Unpacker based on introspection
- Post-processing of dumps
- Increasing compatibility with all packer classes
- Other ways to determine end of monitoring phase

References

Cutler, S.
Malshare.
http://malshare.com/.
Accessed: 2019-02-12.

Plohmann, D., Clauss, M., Enders, S., and Padilla, E.
Malpedia: A collaborative effort to inventorize the malware landscape.
The Journal on Cybercrime and Digital Investigations 3, 1 (2018).

Ugarte-Pedrero, X., Balzarotti, D., Santos, I., and Bringas, P. G.
Sok: deep packer inspection: a longitudinal study of the complexity of run-time packers.
In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 659–673.

Presentation agenda

Release

- Code is not in a good place, yet
- Please wait for the release on `https://github.com/UrmelAusDemEis/RoAMer`
- thorsten.jenke@fkie.fraunhofer.de
- Thank you for your kind attention.