SysTaint: assisting reversing of malicious network communications

Master Thesis - Gabriele Viglianisi
Matr. no. 836130

Supervisor: Prof. Stefano Zanero
Co-supervisor: Dr. Andrea Continella

Problem: Malware proliferation

- Malware became profitable for criminals
- 63.4 million new samples in Q4 2017*
- It is necessary to employ automated methods to analyze malware

* from McAfee Threat Report March 2018

Sandbox: analyzing known and unknown malware automatically

- Checks the sample against a list of known signatures
- Runs the sample in a controlled environment, logging what it does
- Analyzes the logged information for signs of malicious activities

(Nowadays, most antivirus software employ this approach)

Studying how malware works

- Necessary to
  - develop countermeasures
  - uncover behaviors the sample did not show during the analysis
  - understand how a malware communicates with malicious actors
- Unfortunately mostly a manual process

How to study a malware

- Obtain and read the disassembled code of the malware
- Extract information on the behavior of a sample's code observing it while it's executed
  - Running the sample in a debugger
  - Running the sample with added instrumentation

Challenge: Network communications

- Malware samples communicate with malicious actors
- The analysis depends on external servers, on their response and timing
- Debugging and instrumentation techniques may interrupt network communication

Record-replay solutions can help

- Record the activities we want to study
- Replay it deterministically, extracting the information we are interested in

We leverage PANDA [1] to record the execution of a sample, then extract rich information on its inner workings.

[1] https://github.com/panda-re/panda

SysTaint: Approach

We can semi-automatically extract, from the recorded execution, data that allows the analyst to quickly answer the following questions:

- Which part of the sample's code is responsible for a given behavior?
- How does the sample use the data it acquires?
- What data is the sample exchanging over the network and how is it processed?

Comparison with existing technologies

Commonly used solutions:

- **Sandbox software:** only extract information on the interactions between the sample and the environment.
- **Debuggers:** difficult to use. Require re-executing the malware (and repeating the network communications) multiple times.

Research solutions:

- **Dispatcher:** specific to protocol reverse engineering on malware. We employ similar techniques, but use record-replay and offer a generic and practical analysis tool
- **QIRA:** is a generic record-replay-based reverse engineering tool. It only targets Linux processes and short executions.

Integration with existing sandbox

Recording the execution of the sample does not impact its execution, it is thus possible to

- automatically record the malware being analyzed by a sandbox software
- take advantage of the instrumentation that the sandbox software already uses to detect the use of known system libraries
- enrich its reports, so that the analyst can use SysTaint to inspect the internal functions of the malware related to some given interactions reported by the sandbox software

The integration with existing sandbox software is optional, so that it is possible to use SysTaint to study samples that the sandbox software is not able to analyze.

SysTaint: approach

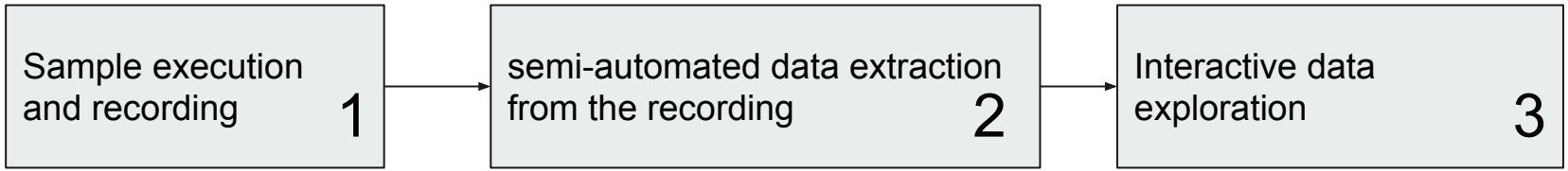
By applying these techniques

- Process memory map extraction
- Cryptographic functions detection
- System and function call tracking
- Data-flow analysis

We can easily extract:

- The processes of interest
- The malware code
- The data the sample exchanges with the system and the network, its provenance or usage
- The unencrypted contents of such data
- The functions that transforms or use the data of interest and their context

SysTaint: approach

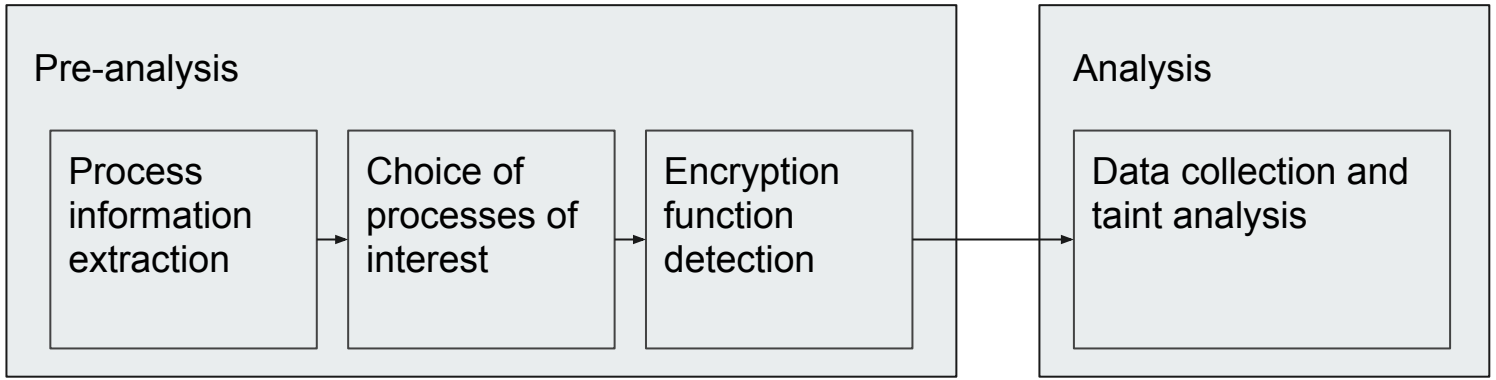


A recording of the sample is obtained

We run analyses extracting useful data from the recording

The analyst queries the collected data and gradually builds an understanding of the malware

Automated data extraction



Techniques: VMI

Consists in extracting information on the processes running on the controlled environment, by observing it from the outside. It allows:

- identifying the memory portions holding the malware code
- obtaining the address of functions in loaded libraries

Techniques: cryptographic functions detection

- Performed through heuristics
- In some cases allows the analyst to immediately locate the encryption functions
  - Extracting the unencrypted data
  - Tracking their output

Techniques: Taint Analysis

- Used to track how some specific data is transformed and used inside a program
- Requires heavyweight instrumentation
- To keep the memory usage low, we track the data the process obtains from the system (e.g., files, the registry, and the network)
- We use the information obtained via taint analysis to
  - find the provenance of the data the sample writes back into the system,
  - the data processed by encryption function
  - the functions processing data with a given provenance.

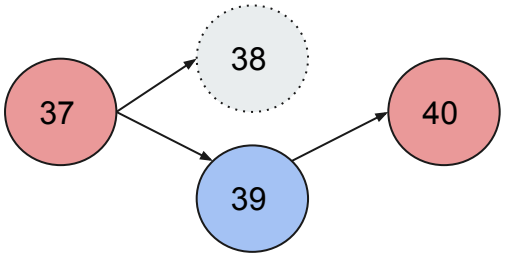
Techniques: Data collection and taint Analysis

We gather data about the internal malware functions in the following manner:

- On each thread, monitor periods of time (events) when the sample is performing a system call or an encryption operation. All data read/written by that thread during that time is tracked via taint analysis and attributed to that event.
- Functions processing tracked data are logged and their input is annotated with the event of provenance

Taint analysis example

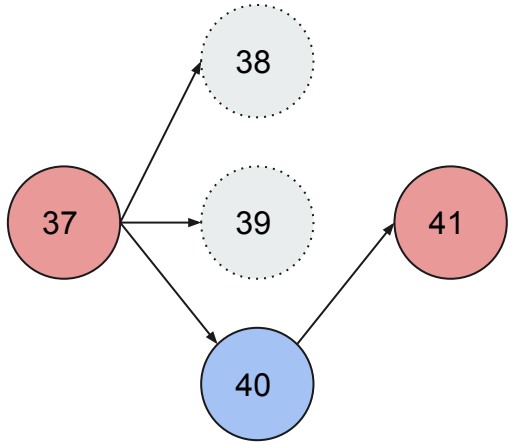
id	description	taint changes	taint labels read
37	syscall(NtReadFile) reads file contents into buffer A	taint[A] <- 37	none
38	call(memcpy) copies A into B	taint[B] <- taint[A]	37
39	call(encryptionFn) encrypts B	taint[B] <- 39	37
40	syscall(NtSetValueKey) writes B into the Windows registry		39



The sample is writing to the registry (40), the contents it has read from a file (37), then encrypted (38)

A more complex example

id	description
37	syscall(NtReadFile) reads file contents into buffer A
38	call(encodingFn) transforms A, puts result in B
39	call(obfuscationFn) of B
40	call(encryptionFn) encrypts B
41	syscall(NtSetValueKey) writes B into the Windows registry



Output of taint analysis

We can locate 38 and 39 by:

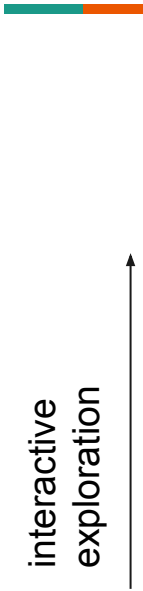
- Looking for functions depending on 37
- Looking for functions reading or writing B
- Looking for functions producing the input to 40

Testing & Evaluation

We tested SysTaint to study four banking trojan samples, respectively Zeus, Citadel, Dridex, Emotet

For each sample, we were able to:

- Find the low level interactions sending specific data on the network
- Find how this data was encrypted, its unencrypted contents and their provenance



Example: Zeus

- Reads report from file
- Decrypts it via RC4
- De-obfuscates it (XOR)
- Unencrypted data
- Obfuscates it (XOR)
- Encrypts it (RC4)
- Sends it through the network

Example: Dridex

- Reads system info (registry and whoami)
- Unencrypted data
- Encrypts it (custom)
- Encrypts it (TLS)
- Sends it through the network

Conclusions

We believe SysTaint can be useful, easy to deploy with existing sandbox solutions, and can speed up the process of studying malware samples, in particular the ones whose behavior depends on network communications with servers outside our control.

Grazie! Domande?

Performances

sample	instructions	execution time	analysis time
Zeus	32,811 M	10 minutes	7 hours
Citadel	12.414 M	4 minutes	2 hours
Dridex	6,853 M	10 minutes	2 hours
Emotet	21,270 M	4 minutes	4 hours

Main challenges

- Working from a hypervisor’s point of view
- Lack of ground truth when working with malware

Future developments

- Per-instruction resolution
- Full dynamic slicing
- Incorporate knowledge of known Windows APIs

Limitations

- Programs employing virtual machines
- We can’t track the output of all internal functions
- Per-function resolution
- Vulnerable to some evasion techniques

Encryption functions detection: details

- Statistics are collected for the first 5 calls to each function
- Each call is evaluated by an heuristic (majority wins)
- Heuristic:
  - If it handles high-entropy data
  - Has a minimum size
  - Most of time is spent on loops
  - Depth of the call tree from there is shallow
- “Promotion” mechanism to find calls to higher level encryption functions from primitives.