

Towards Logical Specification and Checking of Malicious Capabilities

Andrei Mogage¹ Prof. Dr. Dorel Lucanu¹

¹Faculty of Computer Science
"Alexandru Ioan Cuza University" of Iași

March 2024

About Me

- PhD student since 2019.
- Security Researcher @ Bitdefender since 2016.
- Interested in combining **academic** and **industrial** knowledge and tools.

Presentation Outline

- 1 Motivation
- 2 Tainting-Based Logic
- 3 Implementation
- 4 Real-life Use
- 5 Conclusions

Table of Contents

1 Motivation

2 Tainting-Based Logic

3 Implementation

4 Real-life Use

5 Conclusions

What is this malware capable of?

- Malware analysis, but **quick**.
- Overall picture is good, but what about specific information?

Imprecise text from a sandbox report

“The sample performs **certain actions** over the network. This **can include** connecting to remote hosts or sending and reading data from different domains. Not only that, the sample gathers information about the compromised system that can shape its behavior on the system.”

Concrete Evidence

- Follow the money data.
- Actual results, not just guesses.

Spyware Found a string that may be used as part of an injection method
Found browser information locations related strings

Persistence Installs hooks/patches the running process

Fingerprint Queries process information

Evasive Found a reference to a WMI query string known to be used for VM detection
Input file contains API references not part of its Import Address Table (IAT)

Figure: Uncertain information from a sandbox execution report

Table of Contents

1 Motivation

2 Tainting-Based Logic

3 Implementation

4 Real-life Use

5 Conclusions

Tainting-Based Logic (TBL)

- Based on Linear Temporal Logics on Finite Traces.
- Expressing the actions of an application as events in time.

The syntax of TBL consists of three main categories of sentences

- 1 behavioral patterns, whose instances describe behavioral properties of executions;
- 2 capability patterns, whose instances describe capabilities of the executions;
- 3 rules, which relates behavioral properties and capabilities of an execution.

Operators

- Logical connectors: \neg, \vee, \wedge .
- Borrowed temporal connectors: **only X** (Next).
- New temporal connector: **andthen** - *A andthen B* "A holds, and B will hold at some point in the future".

Examples

Atomic Pattern (Fact)

$Tainted(T)$

Behavioral Pattern

$(TaintedAPI(A) \wedge X(Tainted(T)))$
 $andthen PropToAPI(B, T)$

Capability

$File\ Reading(V_0, V_1)$

Tainted Trace

- A trace of actions performed by the application, decorated through taint analysis.
- Finite trace: $\tau = (\tau_0, \dots, \tau_{n-1})$, where τ_i is a set of instances of atomic patterns, written as $P\langle d_1, \dots, d_k \rangle$.

Examples

```
 $\tau = (\{ \text{TaintedAPI}\langle \text{CreateFile} \rangle, \{ \text{Tainted}\langle T_1 \rangle, \{ \text{TaintedAPI}\langle \text{VirtualAlloc} \rangle, \{ \text{Tainted}\langle T_2 \rangle, \{ \text{PropToAPI}\langle \text{ReadFile}, T_1 \rangle, \{ \text{PropToAPI}\langle \text{CryptEncrypt}, T_2 \rangle \} \} \} \} \} )$ 
```

TBL rules

- Describes a relationship between behavioral properties and a capability.
- **If the behavioral pattern is satisfied by the tainted trace, then the analyzed application denotes the specified capability.**

Examples

$$\begin{aligned} & (TaintedAPI(CreateFile) \wedge X(Tainted(T))) \\ & \text{andthen PropToAPI(ReadFile, } T) \\ & \vdash \\ & FileRead(T) \end{aligned}$$

Table of Contents

- 1 Motivation
- 2 Tainting-Based Logic
- 3 Implementation**
- 4 Real-life Use
- 5 Conclusions

Binary Instrumentation and Taint Analysis

- Binary instrumentation for total control.
- Taint analysis for following sensitive data.
- Binary Instrumentation: Intel PIN vs COBAI.

Specification

■ Configuration

- which APIs or code represents **taint sources**;
- what registers or memory can **propagate taint symbols**;
- which additional data should be collected;
- which new predicates should be introduced during specific events.

■ **Rule** The actual rule, which encodes the capability.

■ **Trace generation** Generated through taint analysis, as instructed by the configuration.

Deobfuscation Rule Example

Configuration (regex code as taint source):

- "xor (.), .+",
- "mov byte ptr
\\[(.+)\\+.+\\],\\1"

If the sequence is executed, a new symbol is introduced and the memory area described by the second captured group (the buffer referenced by mov byte ptr) is also tainted. Hence, two new predicates will be introduced: $Tainted(T)$ and $PropToMem(\\2, SZ, T)$.

Rule:

```

andthen(
  ^ (
    Tainted(X),
    X (
      PropToMem(M, SZ, X)
    )
  ),
  TaintedCodeExecute(M, X)
)
⊢ Payload deobfuscation

```


$$\begin{array}{l}
 \text{andthen} \\
 \wedge(\\
 \quad \text{TaintedAPI}(A), \\
 \quad \text{X}(\\
 \quad \quad \text{Tainted}(X_0) \\
 \quad) \\
), \\
 \wedge(\\
 \quad \text{PropToAPI}(B, X_0), \\
 \quad \text{X}(\\
 \quad \quad \text{TaintedAPI}(B) \\
 \quad), \\
 \quad \text{X}(\\
 \quad \quad \text{X}(\\
 \quad \quad \quad \text{Tainted}(X_1) \\
 \quad \quad) \\
 \quad) \\
), \\
 \wedge(\\
 \quad \text{PropToAPI}(C, X_1), \\
 \quad \text{X}(\\
 \quad \quad \text{TaintedAPI}(C) \\
 \quad), \\
 \quad \text{X}(\\
 \quad \quad \text{X}(\\
 \quad \quad \quad \text{Tainted}(X_2) \\
 \quad \quad) \\
 \quad) \\
 \quad), \\
 \quad \text{PropToAPI}(D, X_2) \\
) \\
 \vdash \text{C2 Communication}
 \end{array}$$

Legend:

- A = InternetOpen (Taint Source)
- B = InternetConnect (Taint Source)
- C = HttpOpenRequest (Taint Source)
- D = HttpSendRequest

Informal Explanation

The program has the capability of making a request to an external server, proved by the rule **C2 Communication(0x00CC0004, 0x00CC0008, 0x00CC000C)**. The arguments are the handle values used for the connection setup and the request itself.

Results for Command & Control Communication

Hash (MD5)*	Analysis (sec.)	Native execution (sec.)
04fde4340cc79cd9e61340d4c1e8ddfb	15	14
222792d2e75782516d653d5cccfcf33b	15	13
3d57934b0143b09e82092896580f90b0	15	14
485b9fd90d99ec5e6683fa4448924da8	15	14
66ddb290df3d510a6001365c3a694de2	16	13
6957d10b51d89a2a52eede8b7c3b8472	17	15
6a7fdab1c7f6c5a5482749be5c4bf1a4	17	15
7be33b01e9cb99c6e23ae3b02f384a2c	17	15
9d418ecc0f3bf45029263b0944236884	16	16
c2764be55336f83a59aa0f63a0b36732	15	14
c830512579b0e08f40bc1791fc10c582	15	17
cfcfb68901ffe513e9f0d76b17d02f96	16	18
e81f857bffd0269d9375b08354de3293	16	14

* Samples for DarkSide Ransomware Family

Table of Contents

- 1 Motivation
- 2 Tainting-Based Logic
- 3 Implementation
- 4 Real-life Use**
- 5 Conclusions

Forensics

- Used in forensics for verification of capabilities.
- Simplified the process when a quick validation was necessary.
- Tested on a small team (3 members) with enthusiastic results.

All hands on deck: Large Language Models

- Experiments with using LLMs for faster creation of rules.
- Inspiration for other capabilities and their possible implementations.
- **Feedback loop**: Our experience \leftrightarrow Inspiration from LLMs.

Prompt Examples (1/3)

Question: Write 3 capabilities of a malware that steal user's information.

Answer:

- 1 Data Exfiltration:** Malware designed to steal user information can effectively extract sensitive data from a compromised system. This could include personal information such as names, addresses, and contact details, as well as login credentials, [...].
- 2 Keylogging:** Some malware is capable of recording every keystroke made on an infected computer, allowing it to capture passwords, PIN numbers, and other sensitive information entered through the keyboard.
- 3 Man-in-the-Middle (MITM) Attacks:** Advanced malware can perform MITM attacks, intercepting and modifying data transmissions between the user and their online services. This includes email, instant messaging, or even online banking sessions.

Prompt Examples (2/3)

Question: Which Windows APIs are necessary for creating a keylogger?

Answer:

- 1 SetWindowsHookEx:** This API is used to set up a low-level keyboard hook, which intercepts and processes keyboard events before they reach the application.
- 2 GetMessageW, TranslateMessageW, DispatchMessageW:** These APIs are used to create a message loop that processes messages from the keyboard hook and stores them for later use.
- 3 GetAsyncKeyState:** This API can be used to retrieve the current state of a specific key, allowing the keylogger to check whether a key is currently being pressed or released.
- 4 CreateThread:** This API is used to create a new thread for handling the keyboard hook functionality, allowing the keylogger to run in the background and not interfere with the user's experience.
- 5 WriteFile:** This API is used to write the logged keyboard input to a file[...]

Prompt Examples (3/3)

Question: Write a C++ snippet for a keylogging functionality.

Answer:

```
LRESULT CALLBACK Hook(int nCode, WPARAM wParam, LPARAM lParam){
    if (nCode >= 0 && wParam == WM_KEYDOWN){
        char key = MapVirtualKey((unsigned short)wParam,
                                MAPVK_VSC_TO_VIRTUAL);

        std::ofstream logFile("keylog.txt", std::ios_base::app);
        logFile << key;
        logFile.close();
    }
    return CallNextHookEx(hHook, nCode, wParam, lParam);
}

int main(){
    hHook = SetWindowsHookEx(WH_KEYBOARD_LL, Hook, NULL, 0);
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)){
        TranslateMessage(&msg);
        DispatchMessageW(&msg);
    }
    UnhookWindowsHook(hHook);
    return 0;
}
```

Creating a rule inspired by the LLM prompt

```
andthen(  
  ^(  
    TaintedAPICond(SetWindowsHookEx, ('int', 0, 'WH_KEYBOARD_LL')),  
    X(  
      Tainted(X0)  
    )  
  ),  
  TaintedAPI(DispatchMessage),  
  ^(  
    TaintedAPI(MapVirtualKey),  
    X(  
      Tainted(X2)  
    )  
  ),  
  PropToAPI(WriteFile, X2)  
)  
└─ Keylogging
```

Table of Contents

- 1 Motivation
- 2 Tainting-Based Logic
- 3 Implementation
- 4 Real-life Use
- 5 Conclusions**

Summary

- Intersection of "ingredients" from academic and industrial: Taint Analysis, Linear Temporal Logics (TBL), and Malware Analysis (the ultimate goal).
- Every action is monitored, formalized and used to create the trace.
- **If the behavioral pattern is satisfied by the tainted trace, then the analyzed application denotes the specified capability.**
- **Promising** real-life results.

Future Plans

- Expanding the current set of rules.
- Defining and implementing a model for using capabilities as behavioral properties, thus "combining rules".
- Automation.

