Discussion and analysis of scalability and automation

The final chapter of this project is dedicated to exploring the work that has been done in relation to scalability and automation. We imagine the end goal of the path we started on with this project to be an almost fully automated Software as a service (SaaS) solution that can be used to detect vulnerabilities on machines that cannot be patched immediately.

To highlight the work needed to reach the final goal, we will first compare function hooking and Event Tracing for Windows (ETW), explore exactly what information can be extracted from a patch, and hereafter use this information to discuss how all of the work done to detect CVE-2021-24086 might be applied in an automated and scalable way to detect other vulnerabilities.

1.1 Comparison of function hooking and ETW

In the previous chapters we have explored the world of Windows telemetry, analyzed a recent and critical vulnerability, and hereafter combined all of the knowledge gained to detect the vulnerability using the explored telemetry sources. In this chapter we will be exploring and discussing how the techniques used to detect CVE-2021-24086 possibly could be used to detect other vulnerabilities. Furthermore, we will also explore if, and possible how, we can automate the whole process to automatically detect exploitation of all recent vulnerabilities given a patch.

To detect CVE-2021-24086 we explored both ETW and function hooking to gather relevant telemetry to discover a vulnerability in Windows. Unfortunately, we discovered that the detection opportunities, at least for CVE-2021-24086 and other vulnerabilities in `tcpip.sys`, are not great with ETW as most telemetry herein is based on entering *known* unintended states. With function hooking however, we get to define what is an unintended state and can therefore tailor our telemetry to our specific detection needs.

If we disregard the fact that ETW telemetry was not sufficient to detect CVE-2021-24086, we still want to compare the two methods in order to give the reader an overview of weaknesses and strengths for each method. The results of this analysis can be seen in 1.1

Function hooking	ETW

Cons	Complex	
	Long development time	
	No tolerance for error	Unable to customize events
	Might slow down the computer	Not able to create custom events
	Kernel mode	Lack of events
		Unable to detect CVE-2021-24086
Pros	Custom event filtering	
	Customize detection to specfic vulnerabilities	Built-in to Windows
	High level of control	Robust
	Able to detect CVE-2021-24086	Fast
		User mode

Table 1.1: Comparison of function hooking and ETW

As function hooking and ETW is so fundamentally different, it is hard to compare them in a proper way. Function hooking has a high level of customization which also brings a high level of complexity. ETW, while still being somewhat complex, works out of the box, has a high level of robustness, but gives very little in terms of customization. Furthermore, ETW could only be used to detect CVE-2021-24086 after the patch was applied, which is not that valuable in regards to the scope of this project. Given more vulnerabilities, we do believe that ETW could show its worth, but it is impossible to say without analyzing more vulnerabilities. Perhaps a combination of function hooking and ETW is optimal, as both can be run at the same time without large penalties, both in regards to performance, security and scalability.

1.2 Patch information

In **??** (**??**) we analyzed and reproduced a vulnerability using only publicly available information and information gained from the patch. While publicly information was very scarce, extracting information from the patch gave us very valuable and usable information quickly revealing the root-cause of the vulnerability. While developing a full fledged Proof of Concept (PoC) took a lot of time, revealing the root-cause of the vulnerability was very quick.

The goal of this project is mainly to discover the information given by a patch, and determine if this information can be used to detect the vulnerability in an unpatched system. Therefore, we must detail exactly what information can be extracted from a patch given the right tools such as BinDiff[2] and IDA Pro[1]:

1. Very precise byte by byte of the binary changes

2. Which functions were modified, including confidence and similarity analysis

3. Specific addresses where code was modified, added or removed

4. Changes on the assembly level

5. Additions to the binary

Luckily for us, all security patches are distributed separately from feature upgrades, allowing us to only look at changes we know are related to security.

To relate the patch information to detection, it gave us the knowledge of what function triggers the vulnerability (`Ipv6pReassembleDatagram`), the assembly comparison added to fix the vulnerability, and the added ETW telemetry to detect the now known unintended state. To detect CVE-2021-24086 without a patch or a reliable PoC, would ultimately require us to rediscover the vulnerability based purely on publicly available information. We deem this infeasible for us, and most likely many other security professionals, and certainly deem it infeasible to do on a larger scale.

When patch information is analyzed by someone with the right skills within vulnerability research, it gives tremendous value both in detection purposes, but definitely also within exploitation purposes. The question remains however, whether the right information can be gathered programmatically and in a way that is easy to scale and automate. The next sections will explore this further.

1.3 Scalability and automation

1.4 Future work