Reverse engineering of Windows drivers - A state of the art

Anonymized

2nd Jean SEVAUX
CyberDéfense - ENSIBS
Vannes, France
sevaux.e1800010

3rd Erwan FONDIN
CyberDéfense - ENSIBS
Vannes, France
fondin.e2101282

Anonymized

***Abstract*—This research report seeks to highlight the major risk posed by third-party drivers installed on the Microsoft Windows operating system. In particular, the paper will focus on device drivers, which are present on all computers, both personal and professional. Indeed, a flawed development of such software components poses serious problems, which Microsoft is trying to address, for example with the Windows Driver Model (WDM), the Windows Driver Framework (WDF) and Driver Verifier.**
In response to these concerns, this study explores the landscape of reverse engineering methodologies for Windows drivers. It questions the effectiveness of Microsoft's security measures for drivers and the Windows kernel and investigates existing reverse engineering automatic tools and techniques.
The starting point for this research is Popkorn, a tool developed in parallel with a research paper [4], which automates the reverse engineering of WDM drivers and the search for vulnerabilities in critical functions, particularly those manipulating user-supplied data. This study will use Popkron as a basis for further investigation of device drivers' vulnerabilities, expanding the research to WDF drivers.

***Index Terms*—reverse engineering, methodologies, security, drivers, Microsoft, Windows, kernel**

I. Introduction

In the malware industry, attackers strive to gain ever greater privileges on the victim's system, while remaining as discreet as possible. Fortunately for them, a critical component of the operating system can enable them to do just that, it's the kernel drivers. This document presents the critical role of device drivers in modern operating systems and the potential security risks associated with their development. Device drivers serve as vital intermediaries between applications and hardware components, facilitating functions like GPU access, BIOS updates, and network card management in the Windows operating system. Due to the privileged nature of these operations, drivers are loaded into the kernel at the highest privilege level.

Drivers are created by device vendors using Microsoft's frameworks like the Windows Driver Model (WDM) and the Windows Driver Framework (WDF). Unfortunately, some vendors lack rigorous, security-focused development processes, leading to vulnerabilities that can be exploited to gain elevated privileges. To address these concerns, Microsoft requires drivers to be signed by the Windows Hardware Quality Labs (WHQL) and undergo Extended Validation (EV) certification. However, this certification doesn't involve thorough code verification, relying on vendors to ensure their drivers' security adequately. Unfortunately, even certified and signed drivers are not necessarily free of vulnerabilities. Finally, a signed driver is not necessarily a proof of security, as attackers can gain access to stolen certificates [12].

In light of the potential risks associated with drivers, it's essential to consider the broader context of reverse engineering. This document follows an in-depth scientific study of reverse engineering methodologies. The current landscape of reverse engineering will be discussed, outlining the motivations for this project and exploring various research issues in Section II and Section IV, respectively. Section IV delves into the state of the art, positioning our work in the existing landscape, while Section V explains our group's organization. Section VI then presents the use of Popkorn, a tool reversing WDM drivers, at the base of our work presented in Section VII.

II. Background

A. Reverse engineering

First of all, the reverse engineering of a software component is a process of analysing the operational version of this component in order to reconstruct its technical and functional specifications. Reverse engineering is used, among other things, to re-document, convert, maintain or update old applications.

B. Windows kernel driver

A driver is a specialized piece of software that acts as an intermediary between the operating system and hardware components. These drivers enable the OS to communicate with various hardware devices, such as graphics cards, network adapters, and storage controllers. They provide a unified interface that applications running in user mode can utilize to access the underlying hardware without needing to understand the specifics of each device. Kernel drivers in Windows bridge the gap between user-mode applications and the kernel itself.

The kernel in Windows refers to the core component of the operating system responsible for managing system resources and providing essential services to applications and

other system components. It operates at the most privileged level and plays a central role in maintaining system stability, security, and resource allocation. The kernel handles tasks like process management, memory management, and I/O operations.

In order to standardize and define Windows drivers development, Microsoft published the Windows Driver Model (WDM) [16]. By following this standard, third-party drivers' developpers were sure that their code would be compatible with all Microsoft Windows operating systems. Today, the standard has evolved, as Microsoft has developed a framework for kernel driver development, incorporating certain security measures: the Windows Driver Framework (WDF) [14]. To emphasize briefly WDM and WDF differences, WDM drivers interact directly with the operating system and are trusted kernel-mode components, which means the system provides limited checks on driver input. On the other hand, the WDF model focuses on the driver's requirements, and the framework library handles the majority of the interactions with the system, intercepts I/O requests, takes default actions where appropriate, and invokes the driver's callbacks as required. The WDF model is object-based and event-driven, providing a more structured and secure approach compared to WDM drivers.

III. Subject's choice motivation

Our team is made up of cyber defence engineers. In this context, it was natural for us to turn our attention to a major flaw in operating systems. We combined the knowledge and skills of each of us, namely reverse engineering and tool development, to look at the security of Windows drivers. Research on this subject is crucial, and is of interest to experts such as the DGA-MI, whose work brings them into contact with it.

Drivers are essential components for the smooth running of all information systems. Without them, communication between devices, other components and the operating system would be impossible. Drivers are therefore present on almost all information systems, so it is imperative to ensure that they are secure. One of the reasons for taking an interest in the reverse engineering of peripheral drivers is the common and specific structure shared by many drivers, particularly those close to the kernel. This characteristic suggests that potential security flaws could also be shared between different drivers, thereby increasing the scope of vulnerabilities.

IV. State of the art

Recent developments in the cybersecurity realm highlight the importance of understanding and mitigating risks. A group of hackers, known as SpyBot, has released a tool on a Russian forum that can potentially disable any anti-virus or threat detection and response (EDR/XDR) software. This tool employs a technique referred to as "Bring Your Own Vulnerable Driver" (BYOVD), which involves introducing a legitimate signed driver onto the victim's computer. This driver can then be exploited to undermine the victim's security solutions, as detailed in [17]. These incidents underscore the critical need for robust security measures in both hardware and software development.

In this survey that covers the tools and methods used for vulnerability research on drivers, the following questions are addressed :

- Is the current automation of vulnerability research effectively developed?
- What can be its limitations?

A. Standards for driver development

Drivers are complex objects capable of executing code with kernel privileges. In an attempt to protect Windows users, Microsoft has devised two models on which to base driver development: WDM (Windows Driver Model) and WDF (Windows Driver Framework). Manufacturers, who own their own drivers, must follow one of these two models in order to be validated. This validation involves WHQL (Windows Hardware Quality Labs), the laboratory in charge of certifying the quality of drivers intended to run on Windows. Unfortunately, this verification is not enough to protect Microsoft's operating system. Various techniques involving reverse engineering and vulnerability scanning have highlighted the weaknesses of this certification method.

B. Reverse engineering and analysis tools and methods

Nowadays, there are numerous tools available for reversing various kinds of software, whether it's drivers, executables, DLLs, ... This project focuses on the analysis of drivers in order to find vulnerabilities or configuration flaws within it. After reading the scientific literature, these tools can be classified into four categories. It will help to find out what tools can be used for this project :

1) The first type of tool are well-known pure reverse engineering tools such as IDA, Ghidra and angr. These tools are excellent for reverse engineering but do not easily provide added value for automated vulnerability research on their own. IDA Pro could be used for scripting, but a as a research project, something reproducible is preferable, so proprietary tools are not in the scope. However, Ghidra or Angr can be used for manual verification of an automated method. These two tools are open-source, and a script can interface with them.

2) The second type of tool is turnkey automated analysis tools that use reverse engineering. Those are often open source. The scientific literature gives us some interesting examples such as POPKORN [4]. It is a light- weight framework that harnesses the power of taint analysis and targeted symbolic execution to automatically find security bugs in Windows kernel drivers at scale. Moreover, it detects when unsanitized

user input (the source) can reach functions that provide access to critical kernel resources (the sinks). This tool comes with a specific method of analysis explain before, and it seems to be one of the best tools found so far. Other researchers do not work in an academic context and communicate their work in blog articles. One in particular, Voidsec (a.k.a. Paolo Stagno), publishes its reverse engineering tools on an open-source Github. The IDA databases for each of he's public analyses can also be found there. [17]. Another tool called RevNIC reverses a driver's logic to reproduce it and load it into a chosen operating system and then tests its security [3]. It's a unique tool but it's not open-source.

3) The third type of tool is an implementation of a security qualification procedure owned by private companies like Microsoft's Static Driver Verifier tool. Popkorn was created in response to the inefficiency of this tool [4]. In fact, SDV only supports a set of specific assertions, rules, and tests that focus on the correct usage of Windows APIs. Consequently, the applicability of these tools is restricted to a limited subset of security vulnerabilities [4]. Another paper studied SDV and said that they tested the tool and found 12 false positives on 65 coding errors within drivers. They also found some lack of analysis within this tool and improved it [1].

4) The forth type of tool are fuzzers. They are used for analysis of driver behavior such as IOCTL Fuzzer. This one dynamically captures all DeviceIoControl requests to the testing driver by hooking the NtDeviceIoControlFile function in kernel mode [9]. It could be compared to a Wireshark for driver. But this tool do not cover all DeviceIoControlCodes for drivers using dynamic allocated memory. He can't pass through an anti-reverse system and the tool must be upgraded for being more accurate and powerful to deal with this problem.

C. Critical analysis

All the methods and tools mentioned above seem promising and are showing conclusive results. Most of them come with turnkey tools. Next task is to be able to test and appropriate these tools, starting with the most recent: Popkorn. However, according to the creators of Popkorn, their tool is not perfect. In fact, like all the tools presented, it only focuses on drivers that follow the Windows Driver Model and not the Windows Driver Framework. The latter is the model recommended by Microsoft, but porting drivers from one model to another can be a slow process. Finally, Popkorn was developed for the release of the research paper and has not been updated since.

Finally, note that the automation of reverse engineering to search for vulnerabilities is a subject that is gaining in importance over the years. Numerous tools are being produced and formal methods are emerging. On the other hand, the issue of drivers is complex and there is still a long way to go to achieve "complete code coverage".

V. Group organisation

For this project, - ---------- was chosen as the project leader, as she has a more solid experience in reverse engineering and came up with the idea for the project in the first place. Jean has been appointed as internal and external relations and communications manager. Finally, --- ------ and ---------- are both part of the project team. Everyone is involved in the successful development of the project and have the same technical skills. As for time management, we set up a Gantt chart to give us an overall view of the project's progress. We also set up a time-tracking file showing our estimated and actual worked hours.

VI. Popkorn

Taking Popkorn as a starting point, the solution analysis will be conducted as follows :

1) Exact reproduction of the results presented in the research paper: this will validate the tool's operation as presented by its developers;
2) Update the test environment: make sure the tool isn't obsolete, and repair it if necessary;
3) Update the driver dataset: select new drivers to analyze for the research;

A. Reproduction of the initial POC

Popkorn's developpers supplied their code and dataset on Github, so it's mainly open-source. However, they used a downloader to gather a significative amount of drivers from softwares on the Internet and did not provide it. For the purposes of this paper, the aim was only to recover the metrics given, so the 271 drivers supplied (out of X) were sufficient. The table below shows the results obtained when the test was reproduced exactly.

	Original test	New identical test
Number of drivers used	212	271
Unique bugs found	38	37

First of all, the driver dataset does not include 212 drivers as announced by the researchers in their research paper, but 271. Moreover, among these 271 drivers, the tool qualified 37 vulnerable drivers, not 38. However, the number of vulnerabilities found in these 37 drivers is not specified, so it's possible that several vulnerabilities are present in the same driver. The tool showed 60 timeouts on drivers, hindering the analysis of 60 drivers for the final metrics.

B. Test environment update

Although the results are slightly different from those expected, the tool as presented in 2022 actually works. It is now appropriate to update the entire test environment to ensure that Popkorn is still functional, a year after its development. The table below shows the results obtained with the following technical updates :

- Ubuntu updated from 20.04 to 22.04 ;
- Python updated from 3.8 to 3.12 ;
- virtualwrapper replaced by python3.12-venv ;
- Python libraries updated (angr, ipython, ipdb) ;

	Original test	New updated test
Number of drivers used	212	271
Unique bugs found	38	38

C. Creation of a new dataset

This part of the Popkorn analysis is still in progress, as the WDF drivers are being examined in greater depth. To do this, Popkorn can be re-run on WDF drivers to see how it behaves, although it's not suitable at the moment. WDM and WDF drivers need to be differenciated so that Popkorn's code can be adapted to suit this research. To do this, a number of WDF drivers will be collected, if possible device drivers, and why not their WDM version if it exists, in order to compare results.

For the time being, all the drivers located in C://Windows/System32/drivers were collected from one of the team computer and gave them to Popkorn. The following table shows the results.

Numbers of drivers	454
No sinks found	141
Timeouts	9
Not vulnerable	304

Looking at the results, it seems that at least 445 drivers out of the 454 ones are safe from the vulnerabilities searched by Popkorn. The 9 remaining ones are the timeouts. But note that Popkorn is meant to search WDM drivers only. It is therefore necessary to identify the WDF drivers to measure the blind spot that Popkorn fails to take into account.

VII. Automated analysis of WDF drivers

A. Detection of WDF drivers

The first step in analyzing WDF drivers is to differentiate them from the others. In Popkorn's source code, a function named "find_driver_type" differentiates WDM drivers by looking for the "IoCreateDevice" API call. The "IoCreateDevice" API is not present in WDF drivers. Instead, the function was modified to find the "WdfVersionBind" API, which serves as a wrapper for the DriverEntry function, to bind the code to the correct version of the WDF library.

Then, the script ran on the new dataset of 454 drivers, using only the modified Popkorn function to differentiate WDM, WDF and other drivers. Out of 454 drivers, 169 are WDM and 131 are WDF (leaving 154 drivers of another type). This mitigates the previous results, as only 169 are WDM drivers, the only type of driver supported by Popkorn at the moment.

```
def find_driver_type(proj):
    iocreatedevice_addr =
    ↪  proj.loader.find_symbol("IoCreateDevice")
    wdfversionbind_addr =
    ↪  proj.loader.find_symbol("WdfVersionBind")
    driver_type = ""
    if iocreatedevice_addr:
        print("Found WDM driver: ",
        ↪  hex(iocreatedevice_addr.rebased_addr))
        #logging.info("Found WDM driver: %s",
        ↪  hex(iocreatedevice_addr.rebased_addr))
        driver_type = "wdm"
    elif wdfversionbind_addr:
        print("Found WDF driver: ",
        ↪  hex(wdfversionbind_addr.rebased_addr))
        #logging.info("Found WDF driver: %s",
        ↪  hex(wdfversionbind_addr.rebased_addr))
        driver_type = "wdf"
    else:
        print("Different driver type detected..")
        #logging.info("Different driver type
        ↪  detected..")
    return driver_type
```

B. Exploration of TAU methodology for vulnerability research

During this research, in octobre 31th 2023, an article was posted on the VMWare blog by Takahiro Haruyama, describing it's new tool named TAU. TAU is a tool based on IDA Pro scripts that's meant to automate the hunt of vulnerabilities on WDM and WDF x64 drivers. Unfortunately, even though it is open-source, TAU is not available for everybody because of the IDA Pro requirement.

However, as Takahiro Haruyama explains it's tool's functions, a clear methodology for hunting vulnerabilities in WDF drivers arises. This methodology is different than the one used by Popkorn for WDM drivers because of the WDF specificities. TAU was also developed to fill the potential gaps in Eclypsium's ScrewedDrivers tool, one of the few to take WDF drivers into account. Indeed, with Popkorn and ScrewedDrivers both using symbolic execution with Angr, they share the same flaws ("path explosions, false negatives and other unknown errors"). According to M. Haruyama, a full automation isn't likely to work by completing Popkorn and, even with TAU, a manual verification is still necessary because of WDF complexity.

With this paper, artifacts with an full explanation of TAU methodology presented in Figure 1 for the detection of MmMapIoSpace vulnerabilities are presented, with an IDA database of stdcdrv64.sys. This Intel WDF driver is used as an example in the VMWare article, with a PoC for firmware deletion on Takahiro Haruyama's github page. The targetted API, MmMapIoSpace, is a function of the Windows Memory Management subsystem. It is used by drivers to access the memory of the devices to which they are linked. If the behavior of this API were modified by a hacker, it could

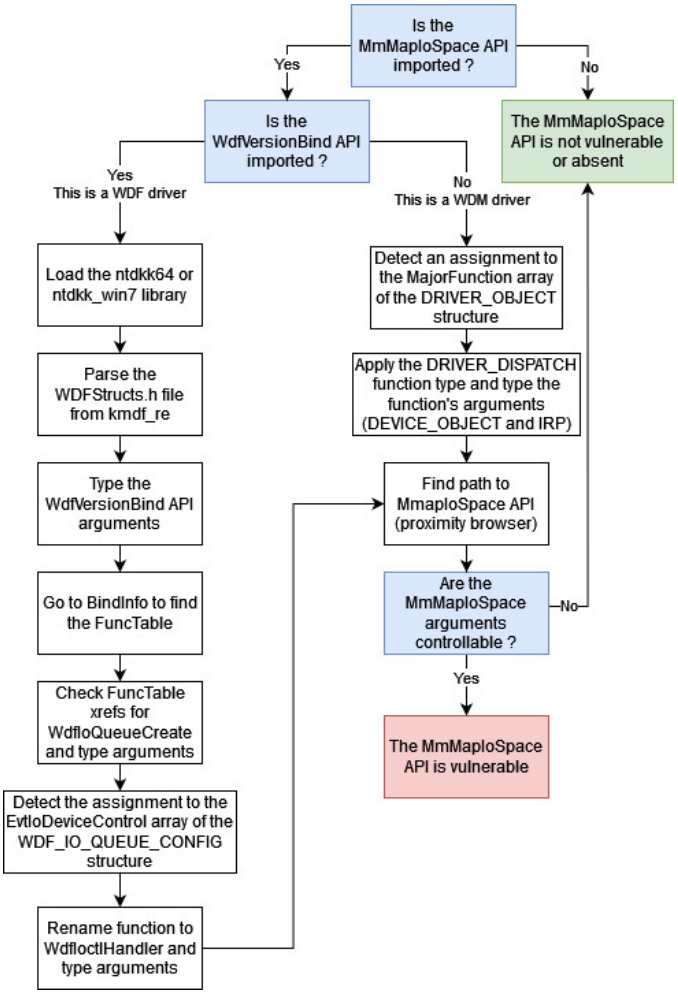

Fig. 1. TAU methodology to detect vulnerable MmMapIoSpace API

access unauthorized memory spaces, cause denial-of-service, execute code, and more.

Those steps were reproduced manually first and detailled in the artifacts, in order to confirm the methodology effectivness. With time, this could be implemented in Popkorn.

VIII. Conclusion

In this paper, the goal was to design an automated reverse-engineering methodology for WDF-type Windows drivers, more specifically for vulnerability scanning. Research is underway for the automation of vulnerability scanning in WDM drivers, but the norm now would be to port drivers to WDF.

To do so, this paper explored the pivotal role that kernel drivers play as intermediaries, granting user-mode applications access to the underlying hardware, often operating at the highest privilege level. After conducting a detailed state of the art, a tool linked to a method published by researchers was chosen, to understand the automated reverse engineering of Windows drivers. This tool, Popkorn, however, was not suited to dealing with WDF drivers, and the only other complete tool discovered to do this was not usable by the general public.

However, this second tool named TAU, a IDA Pro script, was open-source and The next task set is using the taint analysis methodology behind Popkorn to develop a methodology adapted to the search for vulnerabilities in WDF drivers. This research is meant to emphasize the vital need for robust security protocols in the development and assessment of drivers. As it progresses, it is crucial to acknowledge the persistent challenges posed by the ever-changing cybersecurity landscape and work towards fortifying the safeguards surrounding kernel drivers, all while remaining mindful of the wider context of reverse engineering methodologies.

References

[1] Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K. Rajamani, and Abdullah Ustuner. Thorough static analysis of device drivers. *ACM SIGOPS Operating Systems Review*, 40(4):73–85, October 2006.

[2] Vivek Bhardwaj, Vinay Kukreja, Chetan Sharma, Isha Kansal, and Renu Popali. Reverse Engineering-A Method for Analyzing Malicious Code Behavior. In *2021 International Conference on Advances in Computing, Communication, and Control (ICAC3)*, pages 1–5, December 2021.

[3] Vitaly Chipounov and George Candea. Reverse engineering of binary device drivers with RevNIC. In *Proceedings of the 5th European conference on Computer systems*, pages 167–180, Paris France, April 2010. ACM.

[4] Rajat Gupta, Lukas Patrick Dresel, Noah Spahn, Giovanni Vigna, Christopher Kruegel, and Taesoo Kim. POPKORN: Popping Windows Kernel Drivers At Scale. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 854–868, Austin TX USA, December 2022. ACM.

[5] Matt Hand. Methodology for Static Reverse Engineering of Windows Kernel Drivers, May 2023.

[6] Takahiro Haruyama. Vulnerable driver research tool, result and exploit PoCs, October 2023.

[7] Dan Mellinger. Hunting Vulnerable Kernel Drivers, October 2023.

[8] Byungho Min and Vijay Varadharajan. Design, implementation and evaluation of a novel anti-virus parasitic malware. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2127–2133, Salamanca Spain, April 2015. ACM.

[9] Tao Ni, Zhongxu Yin, Qiang Wei, and Qingxian Wang. High-Coverage Security Testing for Windows Kernel Drivers. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, pages 905–908, November 2012. ISSN: 2162-8998.

[10] Enrique Nissim. Reverse Engineering and Bug Hunting on KMDF Drivers.

[11] James S. Okolica and Gilbert L. Peterson. Windows driver memory analysis: A reverse engineering methodology. *Computers & Security*, 30(8):770–779, November 2011.

[12] Denis Pogonin and Igor Korkin. Microsoft Defender Will Be Defended: MemoryRanger Prevents Blinding Windows AV. 2022.

[13] Ariella Robison. Screwed Drivers - Signed, Sealed, Delivered, August 2019.

[14] tedhudek. Kernel-Mode Driver Architecture Design Guide - Windows drivers, March 2022.

[15] tedhudek. Differences Between WDM and WDF - Windows drivers, February 2023.

[16] tedhudek. Windows Driver Frameworks - Windows drivers, February 2023.

[17] VOIDSEC. Windows Drivers Reverse Engineering Methodology, January 2022.