WHAT'S UNDER THE HOOD? A NOVEL VULNERABILITY SCANNER FOR KERNEL DRIVERS

Members:
Kieran Chai Kai Ren, Oliver Yeow Zi Lok
(NUS High School of Mathematics and Science)

Mentors:
Yap Ni (DSO National Laboratories)

Introduction to Problem

- In modern operating systems, **device drivers**[1] are essential to facilitate communication and connections between applications and other hardware components
- Most of these drivers are developed and maintained by **external vendors**
- Current measures **fall short of sufficiently mitigating** most common vulnerabilities
- Legitimately signed drivers are **completely trusted** by security software
- Led to attacks where a vulnerable signed driver is installed into a target system to perform malicious actions (e.g. ransomware)

Aims of Project

1. **Create** a tool that **outperforms** previous state-of-the-art vulnerability scanners for WDM drivers
2. **Find** novel, previously **undiscovered** vulnerabilities with the scanner to improve security of third-party drivers
3. **Optimise** the scanner to improve **speed** of driver scanning

Important Definitions

1. **Privilege escalation** – A malicious user from a low-privilege account, exploiting a vulnerability in the software to gain access to a higher privileged account
2. **Symbolic execution** – Use of symbolic values for inputs in a computer program, to determine the different outcomes that can occur with different inputs
3. **Fuzzing** – Executing the program in an isolated environment, and observing what inputs make it crash
4. **Disassembly** – Converting the machine code of drivers to a more human-readable assembly code
5. **WDM/WDF framework** – Frameworks used for development of Windows Kernel Drivers
6. **IOCTL** – I/O control code, allows for different **functions** to be called in a driver

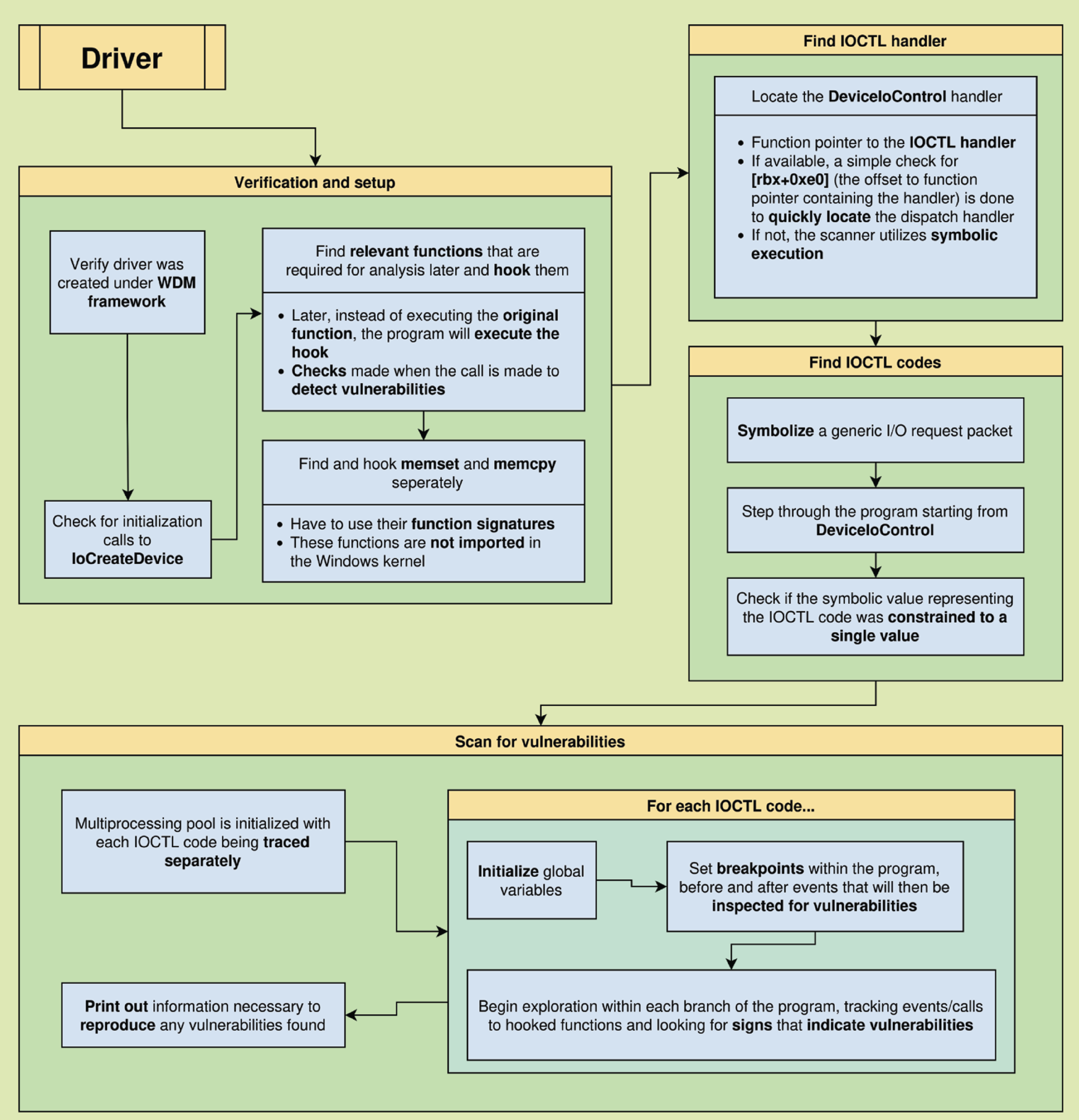
Tools Used

- **Angr** – Library used to perform **symbolic execution** of programs
- **Objdump** – Tool used for **disassembly** of programs
- **Python** – Programming language, used to write the scanner
- **Ghidra** – Tool for **manual analysis** of drivers

Literature Review

- **Static** and **dynamic** verification tools have been developed **by Microsoft**
  - Focused on a set of **specific guidelines** related to the **correct use** of Windows APIs
  - Covers a **limited set** on vulnerabilities
- Existing **third-party tools** that help with **manual analysis**
  - Restrictive
  - Often requires driver's **source code**
  - Or **specific environment** to run the driver on
  - A lot of **work** is still done by the human
- **POPKORN** - A previous attempt to automatically scan for these vulnerabilities
  - Set of several functions **commonly associated** with vulnerabilities (sink functions) **symbolically analysed**
  - Paths to the sinks **traced**, checks if the arguments supplied are loaded from **user-mode buffers** (sources)
  - Lacks **many common vulnerabilities** including vulnerabilities that arise from misuse of **heap functions**
  - **Unable** to scan for vulnerabilities that **do not call** any sink functions

Methodology



Vulnerability analysis – CVE-2018-19320

- Vulnerability in driver **gdrv.sys**
- Caused by insecure IOCTL allowing for **arbitrary memory write**
- Scanner was able to **rapidly** discover and triage the vulnerability **within 25 seconds**

```
DbgPrint("Dest=%x,Src=%x,size=%d",puVar4,lVar5,uVar1);
if (uVar1 != 0) {
    lVar5 = lVar5 - (longlong)puVar4;
    uVar3 = (ulonglong)uVar1;
    do {
        *puVar4 = puVar4[lVar5];
        puVar4 = puVar4 + 1;
        uVar3 = uVar3 - 1;
    } while (uVar3 != 0);
}
uVar2 = 1;
```

Vulnerable code, as shown by **static analysis** performed in Ghidra

```
================================
Reference lpInBuffer:
00000000 00000000
00000000 00000000
00000000 00000000
ARB WRITE DETECT!!!
ADDRESS: <BV64 lpInBuffer[0]>
USERMODE REQUIRED: set()
IOCTL: 0xc3502808
RIP: 0x1400029f7
CALLSTACK:
Backtrace:
Frame 0: 0x140001e81 => 0x1400029b4, sp = 0x7ffffffffeff88
Frame 1: 0x0 => 0x0, sp = 0xffffffffffffffff
CONSTRAINTS:
  Input Buffer: <Bool !((int32)lpInBuffer[4] == 0x0)>
  Input Buffer: <Bool lpInBuffer[0] == 0x0>
================================
```

CVE-2018-19320, as discovered independently by our scanner

Results

- Evaluated on HEVD (Hacksys Extreme Vulnerable Driver)[3], with every vulnerability **accounted for** and reported
- Tested on **physmem_drivers**[2] dataset
- Found **296** vulnerabilities
- Tested on various drivers scraped off the internet
- Discovered and reported 2 **novel** vulnerabilities
- Resulted in 1 **vulnerable** driver being **removed** from download by consumers

Bug Type	Count
Stack overflow	8
Symbolic RDMSR	72
Symbolic WRMSR	72
Arbitrary read	70
Arbitrary write	68
Type confusion	2
Heap overflow	2
Memory disclosure	2
Total	296

Vulnerabilities discovered on physmem_drivers

Conclusion and Discussion

- Scanner can perform **efficiently** and find **many vulnerabilities** quickly in the drivers we obtained
- Rate of **false negatives** may be significantly **higher** in larger, more complex drivers
- While our scanner can detect **standalone** vulnerabilities easily, vulnerabilities that may **chain** into each other and require a complex chain of multiple IOCTLs to trigger **may not be detected** and reported
- **Acceptable** false negative rate for it to be used on a larger scale
- False positives are **extremely rare**
- Due to the nature of symbolic execution, every path in the driver has its full constraints represented and **guarantees** that the path is reachable during execution
- Comparing to existing state-of-the-art, our implementations are also **much faster**, with the average being around **19.45 seconds**, while existing implementations take up to **30 minutes** [4]
- Significant difference between scanning times is likely attributed to **slower** detection methods or **optimisation** methods not being implemented in other tools, which performs analysis with a symbolic IOCTL, preventing use of optimisations such as **multithreading**
- Potentially able to protect many systems from getting exploited by 0-day vulnerabilities by **rapidly discovering and triaging** potential vulnerabilities **before** malicious hackers
- Overall **improvement** upon previous **state-of-the-art** solutions

Future Work

- **Driver Frameworks -** Microsoft is currently promoting the use of **Windows Driver Frameworks (WDF)** to create new drivers. Future work on our scanner could involve additional support to work with WDF
- **Complex Vulnerabilities -** Detection of more **complex vulnerabilities** like type confusions involving heap objects/causing an integer overflow
- **Proof-of-Concepts (PoC) -** Automatically generating PoCs, so that these vulnerabilities can **easily be reported** to the relevant developers to be fixed
- **Patching -** Automated patching system to **update** the drivers and **remove** the vulnerabilities
- **Performance Optimisations -** More work could likely be done to cut the scanning time down, so that the scanner can be used on a **larger scale** to detect many vulnerabilities

Citations

[1] Gillis, A. S., & Tittel, E. (2024, August 5). What is a device driver? Search Enterprise Desktop. https://www.techtarget.com/searchenterprisedesktop/definition/device-driver

[2] Namazso. (n.d.). GitHub - namazso/physmem_drivers: A collection of various vulnerable (mostly physical memory exposing) drivers. GitHub. https://github.com/namazso/physmem_drivers/

[3] Hacksysteam. (n.d.). GitHub - hacksysteam/HackSysExtremeVulnerableDriver: HackSys Extreme Vulnerable Driver (HEVD) - Windows & Linux. GitHub. https://github.com/hacksysteam/HackSysExtremeVulnerableDriver

[4] Zeze-Zeze. (n.d.). GitHub - zeze-zeze/ioctlance: A tool that is used to hunt vulnerabilities in x64 WDM drivers. GitHub. https://github.com/zeze-zeze/ioctlance