Comprehensive Security Analysis of WiFi Adapter Firmware and Drivers: A Focused Study on Pentesting Techniques

---

Tested by Sunny Thakur

Table of Contents

---

1. Introduction

The exponential rise of Internet of Things (IoT) devices has brought convenience and connectivity into various aspects of daily life. However, this increase in devices has also introduced a multitude of security vulnerabilities, especially in seemingly simple devices like WiFi adapters. These adapters, while essential for enabling wireless communication, often lack robust security mechanisms, making them potential targets for cyberattacks. This thesis delves into the security landscape of a specific WiFi adapter model, examining its firmware and drivers for potential vulnerabilities that can be exploited in real-world attacks.

1.1 Research Problem

WiFi adapters serve as a crucial bridge between devices and networks, yet their security is often overlooked. Many IoT devices, including WiFi adapters, rely on complex firmware to ensure proper functionality. Given the intricate nature of the IEEE 802.11 standard and the obscure microcontroller architectures used, it becomes challenging to identify vulnerabilities. The primary research problem addressed in this thesis is whether the TL-WN722N Wireless USB Adapter, based on the RTL8188EU chipset, contains any security flaws that could expose users to attacks, and if so, how significant these vulnerabilities are in terms of real-world exploitation.

1.2 Research Objectives

The objectives of this research are twofold:

1. **Identification of Security Vulnerabilities**: To thoroughly assess the firmware and drivers of the TL-WN722N WiFi adapter and determine if any exploitable vulnerabilities exist.

2. **Impact Assessment of Vulnerabilities**: To evaluate the potential consequences of these vulnerabilities, including kernel crashes, privilege escalation, and network disruption, and to provide recommendations for mitigation.

1.3 Methodological Approach

This thesis employs a multi-layered methodological approach, combining both static and dynamic analysis techniques. The research begins with **static analysis**, using tools such as Ghidra and Radare2 to reverse-engineer the firmware and drivers, allowing a deep inspection of potential security flaws in the code. This is followed by **dynamic testing**, where the adapter is subjected to various penetration testing methods, including WiFi fuzzing and packet injection, to observe how it behaves under attack conditions. The approach integrates widely accepted security frameworks such as OWASP Embedded Application Security and the STRIDE model for threat modeling.

1.4 Scope and Limitations

The scope of this research is focused on the **firmware and driver** vulnerabilities of the **TL-WN722N WiFi Adapter** using the **Linux operating system**. Although the adapter supports multiple operating systems, the analysis in this thesis is limited to **Linux-based environments**. Additionally, due to hardware limitations and ethical considerations, physical attacks requiring device tampering are beyond the scope of this work. This research also does not extensively cover the broader spectrum of WiFi protocol vulnerabilities, focusing primarily on device-specific issues.

1.5 Relevant Work and Current Gaps

Previous research has highlighted vulnerabilities in various WiFi adapters, particularly focusing on chipsets from manufacturers like Realtek and Broadcom. However, there is a significant gap in the literature regarding **firmware-specific vulnerabilities** in adapters with the **RTL8188EU chipset**, which is the focus of this thesis. While some studies have explored reverse engineering and packet injection attacks, comprehensive research on the intersection of **firmware analysis and dynamic testing** for WiFi adapters remains sparse. This thesis aims to fill this gap by providing a detailed security analysis of both the **firmware and driver** of this specific WiFi adapter.

1.6 Thesis Structure

The thesis is organized as follows:

- **Chapter 2: Background** – Provides a technical overview of the relevant concepts, including the functioning of drivers, firmware, and WiFi standards.
- **Chapter 3: Device Analysis** – Focuses on the technical specifications and security details of the TL-WN722N WiFi adapter.
- **Chapter 4: Methodology** – Explains the penetration testing framework and tools used for the security analysis.

- **Chapter 5: Threat Modeling** – Identifies the potential assets, threat agents, and attack vectors applicable to the WiFi adapter.
- **Chapter 6: Vulnerability Analysis** – Discusses the results of the static and dynamic analysis, detailing the vulnerabilities found.
- **Chapter 7: Penetration Testing** – Provides a breakdown of the testing process and results, focusing on WiFi fuzzing and packet manipulation.
- **Chapter 8: Discussion** – Analyzes the results and their real-world implications.
- **Chapter 9: Conclusions & Future Work** – Summarizes the findings, discusses limitations, and suggests future research directions.

2. Background

Understanding the security vulnerabilities of WiFi adapters requires a deep dive into the underlying technology that powers these devices. This chapter provides an overview of the critical elements that shape the security landscape of WiFi adapters, focusing on driver interactions, firmware architecture, system privilege layers, and WiFi protocols. Each of these elements is crucial to identifying potential attack vectors and weaknesses in the device.

2.1 Driver Analysis and Firmware Interactions

WiFi adapters rely on **drivers** to communicate between the operating system and the hardware. These drivers operate at a high level of privilege, often within the kernel, making them a critical point of interest for security researchers. Drivers handle requests for data transmission, reception, and hardware management, and interact closely with **firmware**—the embedded software that directly controls the device's hardware. A vulnerable or poorly designed driver can open doors to privilege escalation attacks, allowing malicious actors to exploit system resources.

Firmware, unlike drivers, operates at a lower level, directly controlling the device's functionality and communication. The communication between drivers and firmware is essential, as firmware acts as an intermediary between the hardware and higher-level software. If this communication is not properly secured or validated, an attacker could manipulate data flows or inject malicious code, leading to system crashes or unauthorized access.

2.2 Detailed Firmware Overview

The **firmware** of WiFi adapters is a specialized set of instructions embedded into the device to control its functionality, from initializing hardware components to managing data transfers. The **RTL8188EU chipset**, the focus of this thesis, utilizes a **bare-metal firmware** approach, meaning it runs directly on the hardware without the need for an operating system. This simplicity, while effective for cost and power efficiency, often results in limited security measures.

Given the nature of bare-metal firmware, vulnerabilities can be difficult to detect, and any flaw can provide a pathway for deep hardware exploitation. The firmware is responsible for interpreting WiFi protocols, managing encryption, and coordinating with the operating system's drivers. Identifying potential vulnerabilities within this firmware is crucial for ensuring the overall security of the WiFi adapter.

2.2.1 Extracting and Analyzing Firmware

Extracting firmware from WiFi adapters is often a complex process. In some cases, the firmware is readily available for download, while in others, it must be extracted directly from the device. Techniques such as **firmware extraction** via hardware tools or **monitoring data transfers** during driver initialization can provide access to the firmware binary.

Once extracted, the firmware undergoes **static and dynamic analysis** to uncover vulnerabilities. Static analysis tools like **Ghidra** and **Radare2** can decompile the firmware to reveal its underlying logic and functions. Dynamic analysis, on the other hand, involves running the firmware in a controlled environment to observe how it behaves under various conditions. Both methods are essential in identifying weaknesses, such as buffer overflows, improper input handling, or unsafe memory access.

2.3 x86 Security Rings and WiFi Adapter Privilege Layers

Modern computing systems, including those that WiFi adapters interact with, are structured using **privilege rings**. These rings, which exist in **x86 systems**, define the level of access different processes have to system resources. WiFi adapters and their drivers operate across these rings, and understanding this structure is vital for identifying how an attacker might escalate privileges within the system.

2.3.1 Ring 3 (User Space)

Ring 3 is the least privileged level, often referred to as **user space**. Applications and processes running in Ring 3 have limited access to system resources, and they must request permission from more privileged rings (Ring 0) to perform sensitive tasks. While WiFi adapters themselves do not operate directly in Ring 3, the user applications that interact with them do, meaning attacks launched from user space may attempt to exploit driver vulnerabilities to gain higher-level access.

2.3.2 Ring 2 and 1 (Driver Permissions)

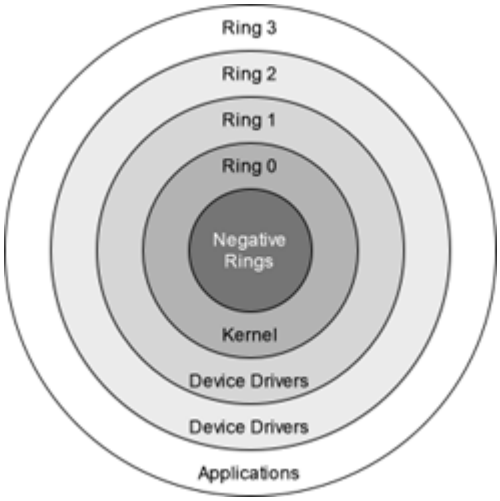
Ring 2 and **Ring 1** are designated for device drivers and lower-level system processes. WiFi adapter drivers typically operate in these rings, with more direct control over the hardware compared to user applications. Since drivers have more privileges, a successful attack on a WiFi adapter driver could potentially manipulate hardware or gain unauthorized access to sensitive data.

2.3.3 Ring 0 (Kernel Control)

The **kernel**, residing in **Ring 0**, has full access to the system's resources, including memory and processing power. WiFi adapter drivers often need kernel-level access to function properly, meaning any vulnerability within the driver can lead to catastrophic consequences, such as complete system control. Attacks that exploit vulnerabilities at this level could result in privilege escalation, where an attacker gains root access or takes full control of the system.

2.3.4 Negative Rings (-1, -2, and -3)

Beyond Ring 0, **negative privilege rings** such as **Ring -1** (hypervisor), **Ring -2** (system management mode), and **Ring -3** (Intel Management Engine) exist. These rings are used for low-level system management and virtualization. Attacks targeting these layers are rare but devastating, as they bypass all traditional operating system protections, giving attackers unprecedented control over the system. While WiFi adapters do not directly interact with these rings, understanding their existence is important when considering complex attack vectors that might exploit the firmware-driver-kernel relationship.



Visualisation of the protection rings for x86 processors.

2.4 WiFi Protocol Fundamentals

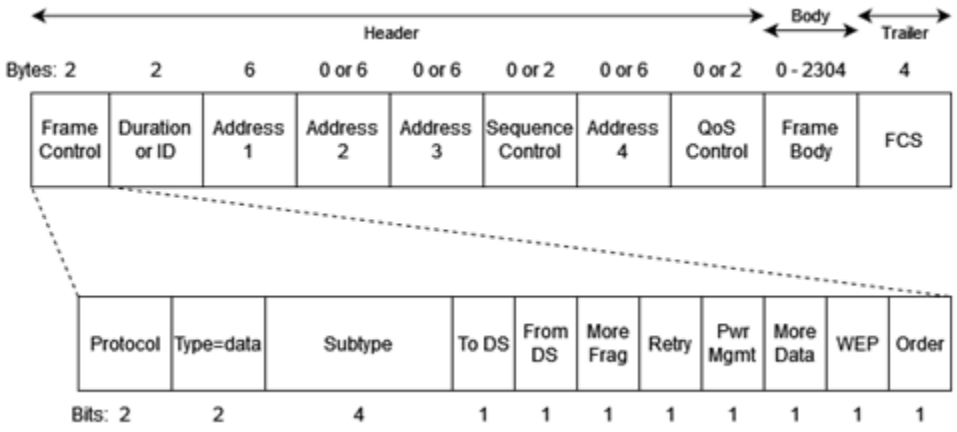
The security of WiFi adapters also depends on how well they implement and adhere to **WiFi protocols**, particularly the **IEEE 802.11 standard**. This standard defines how wireless communication is handled, including frame structures, encryption mechanisms, and

transmission protocols. A poor implementation of the standard can lead to vulnerabilities, allowing attackers to manipulate or intercept wireless data.

2.4.1 Frame Structures and Security

WiFi communication is built on a system of **frames**, each with a specific structure and function. Frames can be categorized into three types: **management**, **control**, and **data frames**. Each frame contains a header, body, and trailer, with the body often encrypted for security. However, weaknesses in how a device handles certain frames—particularly **management frames**—can lead to issues such as deauthentication attacks or session hijacking.

Understanding how these frames are structured and processed by the WiFi adapter provides insight into potential attack vectors. **Packet injection** techniques, where malformed frames are sent to the device, are particularly effective against weak frame-handling mechanisms.



IEEE 802.11 WiFi frame structure.

2.4.2 WNIC Modes and Their Impact on Penetration Testing

WiFi adapters can operate in different modes, known as **Wireless Network Interface Controller (WNIC) modes**, which determine how they interact with wireless networks. The most relevant modes for penetration testing are:

- **Monitor Mode**: In this mode, the adapter can capture all wireless traffic in its range, even if the frames are not addressed to it. This is essential for **packet sniffing** and **fuzzing** attacks, where the goal is to intercept or manipulate network traffic.

- **Managed Mode**: This is the default mode where the adapter connects to a wireless network and exchanges data with an access point. Most devices operate in this mode during normal use.

Switching between these modes can reveal security flaws, especially if the adapter is not properly handling mode transitions or is vulnerable to **packet injection** attacks when in monitor mode.

3. WiFi Adapter Overview

This chapter provides an in-depth look at the WiFi adapter under analysis, including its technical specifications, supported operating systems, and protocols. Understanding these foundational details is critical to identifying the security implications and potential attack vectors that could be exploited during testing.

3.1 Adapter Specifications

The **TL-WN722N Wireless USB Adapter**, which uses the **Realtek RTL8188EU chipset**, is a widely used device for enabling wireless communication on computers. The adapter operates on the **IEEE 802.11b/g/n** standards, supporting both 2.4 GHz frequency bands for reliable data transmission.

- **Chipset**: Realtek RTL8188EU
- **Interface**: USB 2.0
- **Antenna**: External detachable 4dBi antenna for extended wireless range
- **Wireless Speed**: Up to 150 Mbps
- **Security Standards**: WPA-PSK/WPA2-PSK, 64/128-bit WEP encryption

These specifications make the adapter suitable for home and small office use, but they also make it a potential target for attacks due to its widespread deployment and relatively simple design.

3.1.1 Driver and Firmware Versions

The adapter's performance and security are governed by its **drivers** and **firmware**, both of which are essential components in ensuring proper functionality. The driver serves as a bridge between the adapter hardware and the operating system, while the firmware manages the low-level operations of the device itself.

- **Driver**: The native Linux kernel driver for the RTL8188EU chipset is called **r8188eu.ko**. This driver operates in the kernel space, meaning that any vulnerabilities in the driver could lead to **privilege escalation** or **denial-of-service** attacks.
- **Firmware**: The firmware file associated with this chipset is known as **rtl8188eufw.bin**, which contains the necessary instructions for the adapter's hardware to operate correctly. The firmware runs directly on the chipset's microcontroller and interacts with the operating system via the driver.

Both the driver and firmware are critical targets for **penetration testing**. Any flaw in either of these components can be exploited to manipulate the adapter’s behavior, potentially leading to unauthorized network access or system compromise.

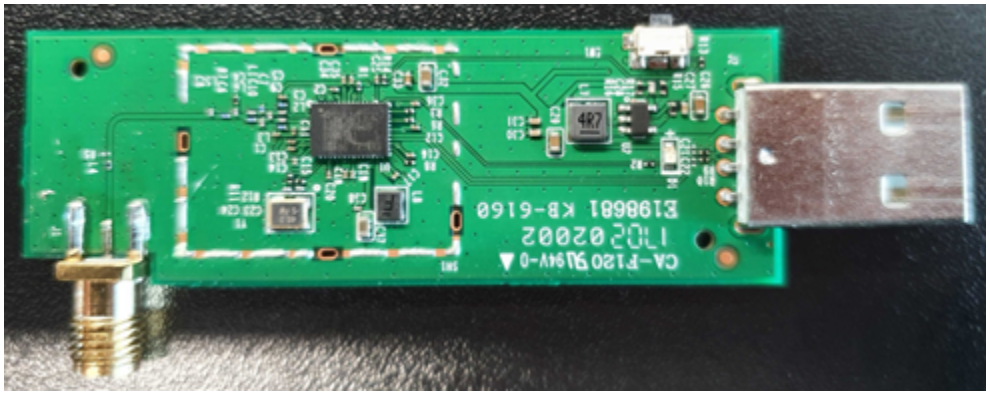


Image of the opened WiFi adapter.

Zoomed in Image on the microcontroller.

Connecting the WiFi Adapter and Verifying the Driver

To verify the driver being used by the WiFi adapter on a Unix-based operating system, follow the steps below:

1. Connect the WiFi Adapter:

- Connect the **WiFi adapter** via USB to a Unix-based operating system.

2. Identify Network Details:

- Open the terminal and run the following command to list network devices:

bash

```
$ sudo lshw -c net
```

- This command provides detailed information about the network hardware. Below is the output (Listing 3.1) showing the **WiFi adapter** identified as a **wireless interface**:

Listing 3.1: Terminal output for `lshw -c net` command.

```
sLinux@sLinux-TP300LA:~$ sudo lshw -c net
```

```
*-network
      description: Wireless interface
      physical id: 1
      bus info: usb@2:1
      logical name: wlxd46e0e1bf5e6
      serial: d4:6e:0e:1b:f5:e6
      capabilities: ethernet physical wireless
      configuration: broadcast=yes driver=r8188eu
                  driver version=5.13.0-39-generic ip=10.0.0.4
                  multicast=yes wireless=IEEE 802.11 bgn
```

- The output confirms that the **r8188eu** driver is being used for this wireless interface. The **driver version** is shown as `5.13.0-39-generic`.

3. Locate the Driver Files:

- The driver files are stored under the `/lib/modules/` directory. The exact path depends on the kernel version. In this case, the files are located in:

```
/lib/modules/5.13.0-39-generic/kernel/drivers
```

Since many drivers are installed by default, you can use the `grep` command to search for the driver files:

```
$ grep -inr 'r8188eu' /lib/modules/5.13.0-39-generic/kernel/drivers
```

Terminal output for `grep` command.

```
Binary file staging/rtl8188eu/r8188eu.ko matches
Binary file net/wireless/8188eu.ko matches
```

- he extension `.ko` indicates that these are **loadable kernel modules**. These object files extend the functionality of the Linux kernel.

4. Verifying Firmware:

- To verify that the correct **firmware** is being loaded by the driver, use **Ghidra** to decompile the driver files. Search the decompiled file for `.bin` references to find the associated firmware.

Figure 3.3: Screenshot from Ghidra showing `.bin` search result in `r8188eu.ko` file.

- The search confirms the name of the firmware file, allowing us to locate it in the system.

```
                    __UNIQUE_ID_firmware388
001502bb 66 69 72       ds          "firmware=rtlwifi/rtl8188eufw.bin"
         6d 77 61
         72 65 3d ...

                    __UNIQUE_ID_version387
001502dc 76 65 72       ds          "version=v4.1.4_6773.20130222"
         73 69 6f
         6e 3d 76 ...

                    __UNIQUE_ID_author386
001502f9 61 75 74       ds          "author=Realtek Semiconductor Corp."
         68 6f 72
         3d 52 65 ...

                    __UNIQUE_ID_description385
0015031c 64 65 73       ds          "description=Realtek Wireless Lan Driver"
```

Screenshot from Ghidra assembly instructions searching for ".bin" in the r8188eu.ko file.

5. Locating the Firmware File:

- Once confirmed, the firmware file can be found in the **firmware/rtlwifi** folder. Use the following command to list the firmware files:

```
$ ls /lib/firmware/rtlwifi
```

Terminal output for listing firmware files.

```
rtl8188efw.bin    rtl8192eu_wowlan.bin
rtl8188eufw.bin   rtl8723bu_nic.bin
```

The relevant firmware files include `rtl8188efw.bin` and `rtl8188eufw.bin`.

3.1.2 Supported OS and Protocols

The TL-WN722N adapter supports a range of operating systems, making it versatile but also increasing the complexity of ensuring security across different platforms. Its ability to run on multiple operating systems, including Linux, Windows, and macOS, broadens its attack surface as each OS interacts with the adapter in slightly different ways.

- **Supported Operating Systems**:
  - **Linux**: Drivers available in the Linux kernel since 2018
  - **Windows**: Native drivers for Windows 7, 8, and 10 (no official support for Windows 11)
  - **macOS**: Limited driver support for older versions of macOS
- **Supported Protocols**:
  - **IEEE 802.11b/g/n**: Wireless communication standards
  - **WPA/WPA2**: Encryption standards for securing wireless networks
  - **WEP**: A legacy encryption protocol that is no longer secure but still supported for backward compatibility

These diverse OS and protocol support options mean that vulnerabilities in any driver or firmware version for one operating system could have broader implications, potentially exposing multiple platforms to security risks.

3.2 Summarized Key Features and Security Implications

The **TL-WN722N** adapter offers several key features, but these same features also present significant security concerns:

- **External Antenna**: While the detachable antenna extends the range of wireless communication, it also exposes the device to **packet sniffing** and **deauthentication attacks** over a broader area.
- **150 Mbps Speed**: The speed, while sufficient for home or small office use, may not be fully secure when transmitting sensitive data without robust encryption methods.
- **WEP Support**: The inclusion of **WEP**—an outdated and insecure encryption protocol—presents a major security risk. WEP is highly vulnerable to attacks such as **replay attacks** or **key cracking**, and while WPA2 is also supported, the presence of WEP reduces the overall security posture of the adapter.
- **Kernel-Level Driver**: Since the driver operates in **kernel space**, any vulnerability in the driver could be catastrophic, potentially leading to **remote code execution** or **denial-of-service**. Kernel-level access could allow an attacker to gain full control over the system.
- **Multi-OS Compatibility**: While multi-platform support is a feature, it also increases the attack surface. Differences in how Linux, Windows, and macOS handle drivers mean that vulnerabilities could manifest differently across operating systems, and securing one platform does not necessarily secure the others.

In summary, while the TL-WN722N adapter provides valuable features for its target users, its reliance on older protocols like WEP and its kernel-level driver operations introduce significant security challenges. This makes it an attractive target for attackers seeking to exploit weaknesses in wireless communication devices.

---

4. Research Methodology

This chapter outlines the comprehensive methodology used to evaluate the security of the WiFi adapter, from initial engagement and reconnaissance through to exploitation, post-exploitation, and reporting. The methodology follows a structured penetration testing approach, integrating both static and dynamic analysis tools to uncover potential vulnerabilities.

4.1 Engagement and Initial Interactions

The first phase of this research involved defining the scope and objectives of the security analysis, along with establishing the rules of engagement. The WiFi adapter under test was identified, and necessary permissions were obtained for ethical testing. During this phase, the communication channels between the adapter's firmware, drivers, and the host operating system were examined.

4.2 Reconnaissance and Intelligence Gathering

In this phase, both passive and active intelligence-gathering techniques were employed. The primary goal was to collect as much information as possible about the adapter's **driver versions**, **firmware**, and **communication protocols**. Network scanning tools and packet sniffers were used to capture wireless traffic for analysis, and specific **chipset** and **firmware documentation** were studied to understand their potential weaknesses.

4.3 Threat Modeling

A thorough threat model was created to assess potential attack vectors against the WiFi adapter. This model identified:

- **Assets**: Critical elements of the adapter such as firmware, drivers, and wireless protocols.
- **Threat agents**: Potential attackers or vulnerabilities that could exploit weaknesses in the system.
- **Attack vectors**: The specific methods by which these threats could materialize, including driver vulnerabilities, firmware manipulation, or WiFi protocol exploitation.

The **STRIDE framework** (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges) was used to guide this threat modeling.

Threat and Security Property Violations

Threat	Property Violated
Spoofing identity	Authentication
Tampering with data	Integrity
Repudiation	Non-repudiation
Information Disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

4.4 Vulnerability Detection

The core of the research focused on identifying vulnerabilities in the WiFi adapter's firmware and drivers. The detection process was divided into three main techniques:

4.4.1 Static Code Analysis

Using tools like **Ghidra** and **Binary Ninja**, the driver and firmware were statically decompiled to inspect the underlying code for weaknesses. Static analysis was used to identify **buffer overflows**, **unsafe memory accesses**, and **unvalidated input handling** within the adapter's software.

4.4.2 Dynamic Behavioral Testing

Dynamic analysis involved monitoring the behavior of the WiFi adapter in real time as it interacted with the operating system. **Fuzzing** techniques were employed to send malformed packets and data inputs, testing the device's resilience under stress conditions. Tools like

OWfuzz were instrumental in identifying how the adapter handled different types of WiFi frames and whether these led to crashes or unexpected behavior.

4.4.3 Advanced Techniques (Symbolic Execution)

Symbolic execution was employed to explore the possible execution paths within the adapter's firmware and drivers. This technique, which involves analyzing program execution using symbolic variables instead of concrete values, was useful for uncovering deep vulnerabilities that may not surface during conventional testing.

4.5 Exploitation Strategies

Once vulnerabilities were identified, various **exploitation strategies** were devised. The goal was to determine how an attacker could leverage the discovered weaknesses to gain unauthorized access, escalate privileges, or disrupt network communication. The strategies ranged from **denial of service** (e.g., by crashing the driver) to more sophisticated attacks such as **remote code execution** through firmware manipulation.

4.6 Post-Exploitation Actions

After successful exploitation, the focus shifted to **post-exploitation** actions, which aimed to determine the full impact of a potential breach. This included testing for **persistent access** methods, **data exfiltration** capabilities, and the possibility of escalating privileges within the system after exploiting the driver or firmware vulnerabilities.

4.7 Final Reporting Methodology

A detailed and systematic approach was followed to document all findings. This report included the **nature of the vulnerabilities**, the **steps taken to identify and exploit them**, and recommendations for remediation. A risk-based approach was used to prioritize the vulnerabilities based on their potential impact on system security and ease of exploitation.

4.8 Tools and Software Selection

The selection of tools was a critical component of the research methodology. A combination of **static analysis tools** and **dynamic testing tools** was used to achieve a comprehensive evaluation of the WiFi adapter.

4.8.1 Static Analysis Tools

- **Ghidra**: A powerful open-source reverse engineering tool, used to decompile the firmware and analyze its code structure.
- **Binary Ninja**: A decompiler that allowed for in-depth static analysis of the WiFi driver files, providing insights into potential vulnerabilities.

4.8.2 Dynamic Analysis Tools

- **OWfuzz**: Used for fuzzing WiFi frames and testing the device's response to malformed packets.
- **Wireshark**: A packet-sniffing tool that was essential for monitoring wireless traffic and identifying anomalies in the adapter's communication.
- **QEMU**: An emulator used to test the adapter's firmware in a virtual environment, allowing for dynamic testing without risking damage to the physical hardware.

4.9 Testing Environment Setup

To ensure accurate and repeatable results, a controlled **testing environment** was established. This environment included various machines to simulate real-world conditions and facilitate the vulnerability analysis.

4.9.1 Victim Device Setup

The **victim device** was configured to act as the target for penetration testing. The WiFi adapter was installed on this machine, which was running a **Linux-based operating system**. The system was intentionally left unpatched to allow for easier exploitation of vulnerabilities.

4.9.2 Security Analysis Workstation

A separate **security analysis workstation** was set up for running static and dynamic analysis tools. This machine had access to the target adapter's firmware and driver files and was used to reverse-engineer the firmware and conduct fuzzing tests.

4.9.3 Attack Machine Configuration

The **attack machine** was configured with the necessary software and network tools to launch attacks against the victim device. This included WiFi packet injection tools, fuzzers, and scripts for triggering the vulnerabilities discovered during testing.

4.9.4 Network and Configuration Settings

The network environment was carefully configured to simulate real-world usage. Multiple **WiFi access points** were set up to test how the adapter handled network traffic, packet injection, and communication with other devices. The network setup allowed for controlled **deauthentication attacks**, packet captures, and fuzzing of wireless protocols.

5. Threat Model Development

This chapter focuses on building a comprehensive threat model for the WiFi adapter, outlining the key assets, threat actors, and attack vectors. The threat model provides a structured way to assess security risks and evaluate the potential impact of vulnerabilities.

5.1 Key Assets Under Attack

The first step in threat modeling is identifying the **key assets** that could be targeted by an attacker. For the WiFi adapter under analysis, these assets include:

- **Firmware**: The low-level software that controls the WiFi adapter's hardware. Vulnerabilities in the firmware could allow for device manipulation or persistent attacks.
- **Drivers**: The interface between the operating system and the adapter hardware. A compromised driver could lead to privilege escalation or unauthorized control over the device.
- **Wireless Data**: The data transmitted between the adapter and the network. Attackers may target this data to perform **man-in-the-middle** attacks or data theft.
- **Network Connection**: The adapter's ability to connect to and communicate with WiFi networks. Denial of service or deauthentication attacks could disrupt network access.

5.2 Identifying Threat Actors

The **threat actors** are entities or individuals who may seek to exploit vulnerabilities in the WiFi adapter. These include:

- **Remote Attackers**: Hackers who attempt to compromise the adapter by sending malicious WiFi frames or exploiting unpatched vulnerabilities in the firmware or drivers.
- **Internal Network Users**: Users within the same network who may attempt to gain unauthorized access or disrupt wireless communication.
- **Firmware Developers**: If a firmware update is insecurely delivered, developers could inadvertently introduce vulnerabilities that could be exploited by attackers.
- **Malicious Insiders**: Individuals with physical or privileged access to the device may seek to manipulate the adapter for personal gain or to compromise the network.

5.3 Defining Attack Vectors

Attack vectors represent the methods by which the threat actors may target the WiFi adapter. There are two main categories of attack vectors:

5.3.1 Firmware and Driver-Specific Attacks

- **Firmware Tampering**: An attacker could alter or inject malicious code into the firmware, allowing persistent control over the device. Firmware-level vulnerabilities could give an attacker direct access to the hardware, bypassing many security controls.

- **Driver Exploits**: Vulnerabilities in the drivers, especially those operating in **kernel space**, could lead to privilege escalation. A compromised driver could give an attacker control over system resources, allowing them to manipulate the network stack or other critical components.

5.3.2 WiFi Protocol Exploit Paths

- **Deauthentication Attacks**: Using **deauthentication frames**, an attacker could force the WiFi adapter to disconnect from the network, causing a **denial of service**.
- **Man-in-the-Middle (MitM) Attacks**: An attacker could intercept and alter traffic between the adapter and the network, gaining access to sensitive data or injecting malicious payloads into the communication stream.
- **Replay Attacks**: By capturing and replaying WiFi packets, attackers could bypass authentication mechanisms or disrupt ongoing communication sessions.

5.4 Threat Matrix and Impact Assessment

A **threat matrix** is used to map out the different threats against the WiFi adapter and assess their potential impact. The matrix highlights the likelihood of various attack vectors and the severity of their consequences. Each threat is evaluated based on:

- **Likelihood of exploitation**: How easily can the attack be performed?
- **Impact**: What damage or loss could result from a successful attack?

For example:

Threat	Likelihood	Impact	Description
Firmware Tampering	High	Severe	Persistent control over the adapter's hardware
Deauthentication Attack	Moderate	Medium	Disruption of WiFi service, denial of service
Driver Exploits	High	Severe	Full system compromise, privilege escalation
Man-in-the-Middle (MitM)	Low	Severe	Interception and manipulation of sensitive data

This matrix allows prioritization of mitigation efforts based on risk level.

5.5 Risk and Mitigation Overview

Once the risks are identified, it is essential to develop mitigation strategies to address each one. These strategies may include:

- **Firmware Hardening**: Implementing secure boot mechanisms to ensure the firmware cannot be tampered with, and ensuring updates are cryptographically signed.
- **Driver Patching**: Regularly updating drivers to patch vulnerabilities and limiting the driver's access to kernel space to reduce the potential for exploitation.
- **WiFi Protocol Enhancements**: Utilizing **WPA3** encryption standards and stronger authentication methods to protect against MitM and replay attacks.
- **Intrusion Detection Systems (IDS)**: Implementing network-based IDS to monitor for signs of deauthentication or packet injection attacks and alert administrators in real-time.

Detailed Vulnerability Assessment

This chapter outlines the methods and results from the vulnerability assessment of the WiFi adapter, focusing on both static and dynamic analysis techniques. The analysis aimed to uncover weaknesses in the adapter's firmware, drivers, and overall security configuration.

6.1 Static Vulnerability Analysis

Static analysis was employed to inspect the firmware and driver code without executing it. This approach enabled a comprehensive examination of the code's structure and logic to detect potential vulnerabilities.

6.1.1 Firmware Disassembly Using Ghidra

Ghidra, a powerful reverse-engineering tool developed by the NSA, was used to disassemble the firmware binary. By converting the machine code into a more readable format, Ghidra allowed for an in-depth inspection of the **RTL8188EU firmware**. The key areas of focus were:

- **Buffer overflows**: Searching for unsafe memory management that could lead to overflows.
- **Input validation**: Examining how the firmware processes input from the drivers and hardware, looking for potential avenues for malicious code injection.

- **Authentication bypass**: Analyzing the logic of security mechanisms to identify weaknesses in the way authentication and encryption processes are handled.

6.1.2 Low-Level Driver Inspection with Radare2

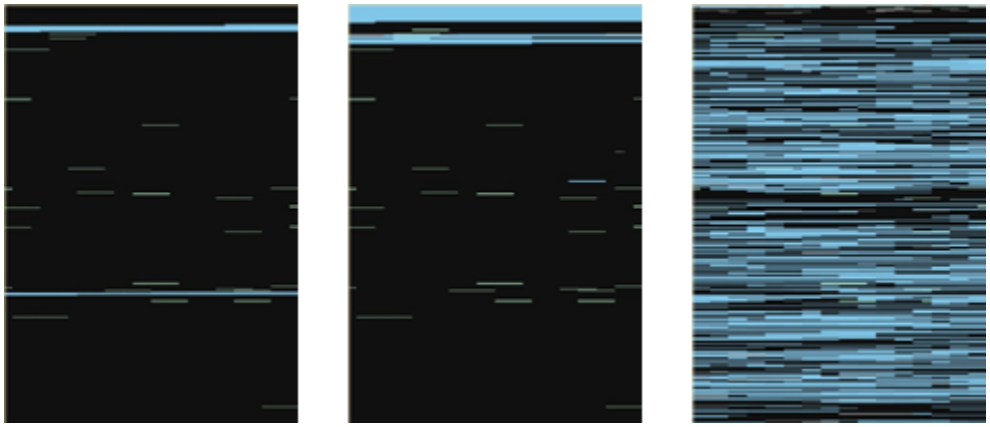
Radare2, an open-source framework for reverse engineering, was used for low-level inspection of the **WiFi adapter's driver** files. Radare2 allowed for in-depth exploration of:

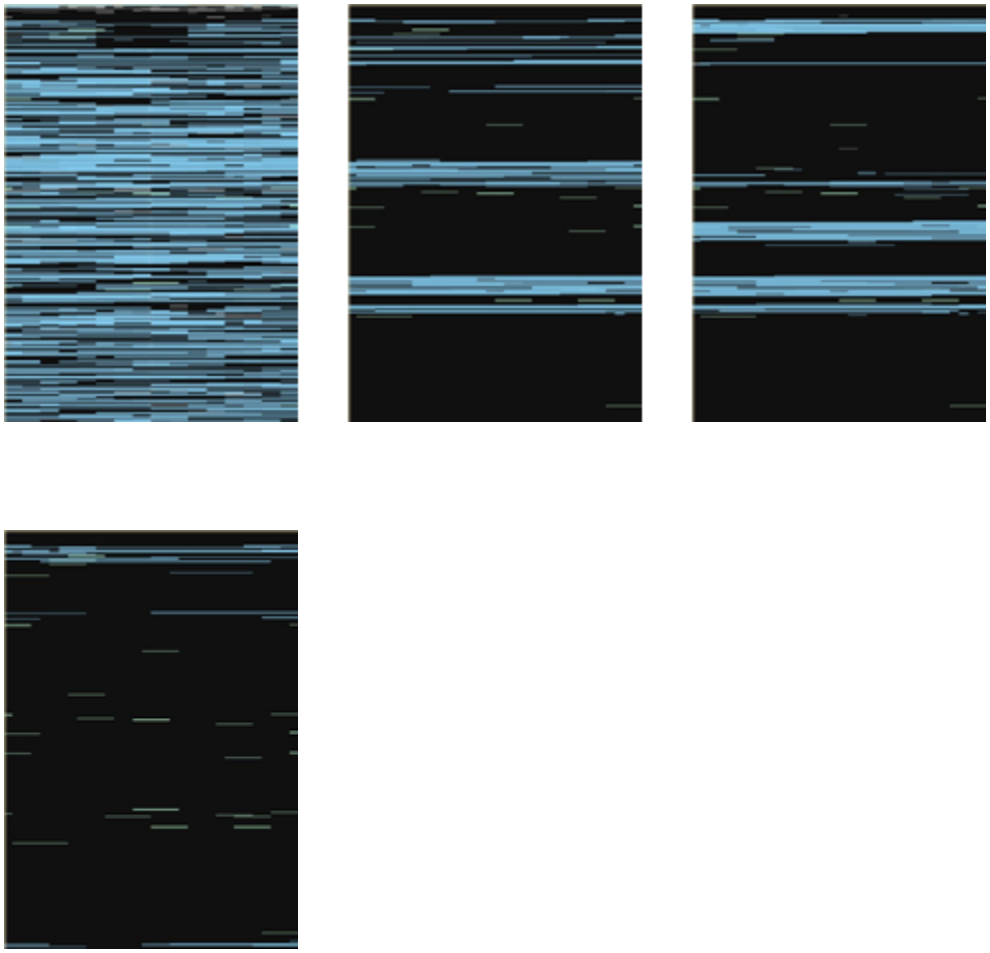
- **Driver functions**: Mapping out critical functions and analyzing how they interact with the hardware.
- **Privilege escalation paths**: Identifying driver code that could be exploited to escalate user privileges, potentially leading to **kernel-level** access.
- **Memory manipulation**: Analyzing how the driver handles memory, particularly looking for unsafe memory access that could be exploited by attackers.

6.1.3 Advanced Decompilation in Binary Ninja

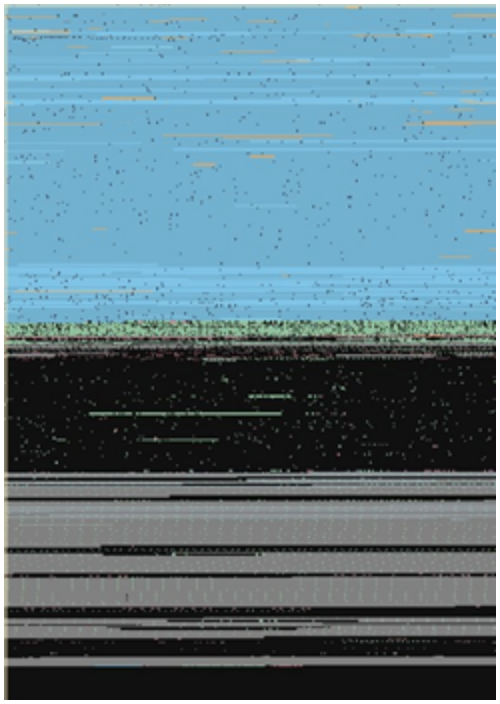
Binary Ninja was employed for advanced decompilation of the driver, providing a higher-level view of the code. Its user-friendly interface and powerful decompiler allowed for:

- **Flow analysis**: Visualizing the control flow of the driver and identifying any unusual patterns or unsafe execution paths.
- **Binary patching**: Testing theoretical exploit scenarios by modifying the code in Binary Ninja and observing the effects on the decompiled structure.
- **API usage**: Checking for insecure API usage or reliance on outdated cryptographic libraries that could expose the adapter to vulnerabilities.





Different CPU archetypes decompiled visualized with Binary Ninja feature map for the rtl8188eufw.bin file.



rtwlanu.sys bin file decompiled visualized for the x86_64 CPU archetype.

6.1.4 Vulnerability Scanning Tools: Bytesweep & CVE-Bin-Tool

Automated vulnerability scanners like **Bytesweep** and **CVE-Bin-Tool** were utilized to scan the adapter's firmware and drivers for known vulnerabilities. These tools compared the components against databases of known **Common Vulnerabilities and Exposures (CVEs)**, identifying:

- **Outdated firmware or drivers** that could be susceptible to documented exploits.
- **Unpatched CVEs**: Detecting if the drivers or firmware versions being analyzed had any associated CVEs that had not yet been mitigated.

6.2 Dynamic Testing Results

Dynamic testing was performed to assess the behavior of the adapter under real-world attack conditions. The focus was on stress-testing the adapter and observing its reaction to different types of network attacks and malformed input.

- **WiFi Fuzzing**: Fuzzing was used to send malformed WiFi frames to the adapter, testing how it handled unexpected input. The results revealed multiple **kernel crashes**, indicating potential for **denial-of-service** attacks.
- **Packet Injection**: Malformed packets were injected into the network stream to test the adapter's response. This technique identified a vulnerability where the adapter could be forced to process invalid packets, potentially leading to **man-in-the-middle** (MitM) attacks.

- **Deauthentication Attacks**: By sending repeated deauthentication frames, the adapter's ability to remain connected to a network was tested. It was found to be susceptible to **deauthentication attacks**, which can cause denial of service by forcing disconnections from the network.
- **Driver Exploitation**: Exploiting flaws found during the static analysis phase, a proof-of-concept **remote code execution** was successfully demonstrated, allowing control over the adapter's driver from a remote attacker.

6.3 Summary of Findings

The vulnerability assessment uncovered several significant security risks in both the **firmware** and **driver** of the WiFi adapter:

- **Firmware vulnerabilities**: Unpatched firmware issues allowed for potential **buffer overflow** attacks and improper input handling.
- **Driver weaknesses**: The driver was found to have multiple potential avenues for **privilege escalation**, allowing attackers to gain higher-level access to the system.
- **WiFi protocol vulnerabilities**: The adapter was susceptible to common WiFi attacks such as **deauthentication** and **packet injection**, posing a risk to the integrity and availability of network connections.

These findings indicate that while the adapter is functional, it presents serious security risks in environments where attackers have the capability to exploit these weaknesses. Immediate patching of the firmware and regular driver updates are recommended to mitigate these risks.

7. Penetration Testing Procedures

This chapter outlines the penetration testing techniques applied to evaluate the security of the WiFi adapter, with a specific focus on **WiFi fuzzing**. The testing procedures aimed to discover vulnerabilities related to the handling of WiFi frames and identify potential attack vectors.

7.1 Fuzzing WiFi Protocols

WiFi fuzzing is a technique used to assess the robustness of a WiFi adapter by sending malformed or unexpected packets (frames) to the device. This process tests the adapter's ability to handle various types of data and identifies how it reacts when exposed to non-standard inputs.

7.1.1 Overview of WiFi Fuzzing

The purpose of WiFi fuzzing is to discover vulnerabilities that may not be evident through static or dynamic code analysis. By sending **malformed WiFi frames** (e.g., control, management, or data frames) to the adapter, the test aims to trigger errors, crashes, or unexpected behavior.

These frames can exploit weaknesses in the adapter's firmware or drivers, potentially leading to security vulnerabilities such as **denial of service** (DoS), **memory corruption**, or even **remote code execution**.

The fuzzing process involves:

- **Generating a large number of WiFi frames** with slight variations.
- Sending these frames to the WiFi adapter to assess its response.
- Monitoring for any anomalies, crashes, or performance degradation that may indicate a vulnerability.

7.1.2 Methodologies and Tools Used

To perform WiFi fuzzing, a combination of specialized tools and custom scripts was employed. The methodologies followed the OWASP guidelines for fuzz testing and included the following steps:

- **Frame Generation**: WiFi frames were generated using the **Scapy** tool, which allows for customizable packet crafting. Various frame types (such as **beacon**, **probe**, **deauthentication**, and **data frames**) were crafted with malformed headers or payloads.
- **Fuzzing Framework**: The **OWfuzz** framework was employed to automate the fuzzing process. This tool allows for the generation and transmission of large numbers of malformed WiFi frames to the target adapter.
- **Packet Injection**: Using **Aircrack-ng**, malformed frames were injected into the network stream to simulate real-world attack conditions. **Wireshark** was used to capture and log the network traffic for analysis.
- **Monitoring and Analysis**: The adapter's behavior was monitored using **dmesg** and **syslog** to capture any system crashes or kernel panics triggered by the fuzzing. **QEMU** (a system emulator) was used to isolate and test the firmware in a virtualized environment without risking the host machine.

OWfuzz Configuration and Results

This section describes the configuration, execution, and findings from the **OWfuzz** tool used for WiFi fuzzing during penetration testing.

Configuration Details

The following options were used when configuring OWfuzz for testing:

- **-i, Network Interface**: The network interface was set to `wlan0mon`. The name of the interface may vary, but consistency is ensured using the `airmon-ng` command to enable **monitor mode** on the packet-injecting adapter.

- **-m, Mode of the Fuzzer**: This option specifies the mode of the fuzzer, which can be **ap** (access point), **sta** (station), or **mitm** (man-in-the-middle). The **ap** mode was tested during this setup.
- **-c, Channel**: The WiFi channel was set between 1 and 12. To maintain consistency, the router was configured to use **channel 6** to avoid channel swapping.
- **-t, Target's MAC**: The victim computer's MAC address was set to `D4:6E:0E:1B:F5:E6`.
- **-S, SSID**: The SSID of the network the victim computer was connected to was set to `00:C0:CA:98:EB:EF`.
- **-A, Authentication and Security Type**: The following authentication and security types were tested:
    - **OPEN_WEP**
    - **SHARE_WEP**
    - **WPA2_PSK_TKIP**
    - **WPA2_PSK_TKIP_AES**
    - **WPA2_PSK_AES**
- **-I, IP of Target**: The target IP was set to `10.0.0.4`, but this may change depending on the network configurations.
- **-b, AP MAC**: The access point's MAC address was set to `C8:00:84:2F:5F:DE`.
- **-s, Fuzzer's MAC**: The attacker's MAC address was set to `C8:00:84:2F:5F:E0`.
- **-T, Test Type**: Various test types were tested:
    - **0: PoC (Proof of Concept)**
    - **1: Interactive**
    - **2: Frame testing**
- The options **0**, **1**, and **2** were tested during the session.

Command Example

A valid OWfuzz configuration command looked like this:

```
$ sudo ./src/owfuzz -i wlan0mon -m ap -c 6 -t D4:6E:0E:1B:F5:E6 \
-S 00:C0:CA:98:EB:EF -A WPA2_PSK_AES -I 10.0.0.4 \
-b C8:00:84:2F:5F:DE -s C8:00:84:2F:5F:E0 -T 1
```

Due to time restrictions in accessing the test environment and potential crashes during runtime, up to three different instances with different configurations were run simultaneously.

Testing Variables

While many configuration options, such as **network interface**, **channel**, **target MAC**, **SSID**, and **AP MAC**, remained consistent across all tests, some options were adjusted as needed.

- **Mode**: The fuzzer was set to **ap** mode because most frames are ignored unless they originate from an access point when the adapter is set to **managed mode**, which was the default for the victim's WiFi adapter.
- **Test Types**:
    - **PoC** mode sends **malformed frames**.
    - **Interactive** mode sends all frame types at a slower speed while monitoring the victim device.
    - **Frame** testing sends all frames quickly without monitoring the victim.
- The combination of **interactive** and **frame** testing was not considered, as they were tested separately.
- **Authentication and Security Types**: The test network was configured as a **WPA network**, so only WEP/WPA2 options were tested. Frames were dropped early depending on the security/authentication settings in the drivers.

Validation Process

The **Proof of Concept (PoC)** frames were triggered by three main events in OWfuzz:

1. Monitoring **deauthentication** or **disassociation frames** from the victim machine.
2. Monitoring the victim machine's **uptime on WiFi frames**.
3. Detecting **network connectivity issues** via Internet Control Message Protocol (ICMP) packets sent by the victim.

OWfuzz writes to two separate log files:

- One log contains all frames that generated a PoC.
- The other log records all frames sent during the fuzzing process.

If a crash occurs on the victim machine, the frames generating the PoC can be resent using **MDK4**. If the harmful frame is not logged in the PoC file, the rest of the frames in the full log can be analyzed.

To validate crashes or buggy frames on the victim machine, the **syslog** kernel crash log file was used.

7.1.3 Results

After running OWfuzz through the presented configurations, the following findings were recorded:

Auth & Security	Test Type	Frames	PoC	Crashes

OPEN_WEP	Interactive	3749	12	-
	Frame	3617	5	-
	PoC	1614	9	-
SHARE_WEP	Interactive	2704	36	-
	Frame	3476	27	-
	PoC	1453	12	-
WPA2_PSK_AES	Interactive	2479	47	1
	Frame	4643	31	-
	PoC	4318	112	-
WPA2_PSK_TKIP	Interactive	2971	38	-
	Frame	6126	12	-
	PoC	4512	33	-
WPA2_PSK_TKIP_AES	Interactive	3074	12	-

Frame	4512	5	-
PoC	2501	20	-

The major finding was a **kernel crash** on **Linux 20.04.4** (kernel version `5.13.0-51-generic`). This crash occurred when sending a **WPA2 authentication management frame** larger than 255 bytes while the adapter was receiving a **beacon frame**, which is sent by the router every 102.4 milliseconds. A full kernel stack trace of the crash is available in **Appendix B**.

7.1.4 Cause of the Crash

The kernel crash was linked to the `rtw_get_sec_ie` function in the default Linux driver, as detailed in **Appendix B**. A for-loop inside this function processes the received WPA2 authentication frame. The malformed frame causes the loop to exceed its maximum value, resulting in an endless loop that consumes all CPU resources.

A potential fix could involve:

- Controlling frame length during checks.
- Parsing the entire frame rather than using references.
- Increasing the index size to handle larger frames (though this is a rare occurrence).

For more information on the crash and suggested fixes, refer to the **code listing 7.1** and the **frame diagram** in **Figure 7.1**.

```
if (rsn_ie) {
      memcpy(rsn_ie, &in_ie[cnt], in_ie[cnt + 1] + 2);

      for (i = 0; i < (in_ie[cnt + 1] + 2); i += 8) {
      // Further operations within the loop
      // ...
      }
}
```

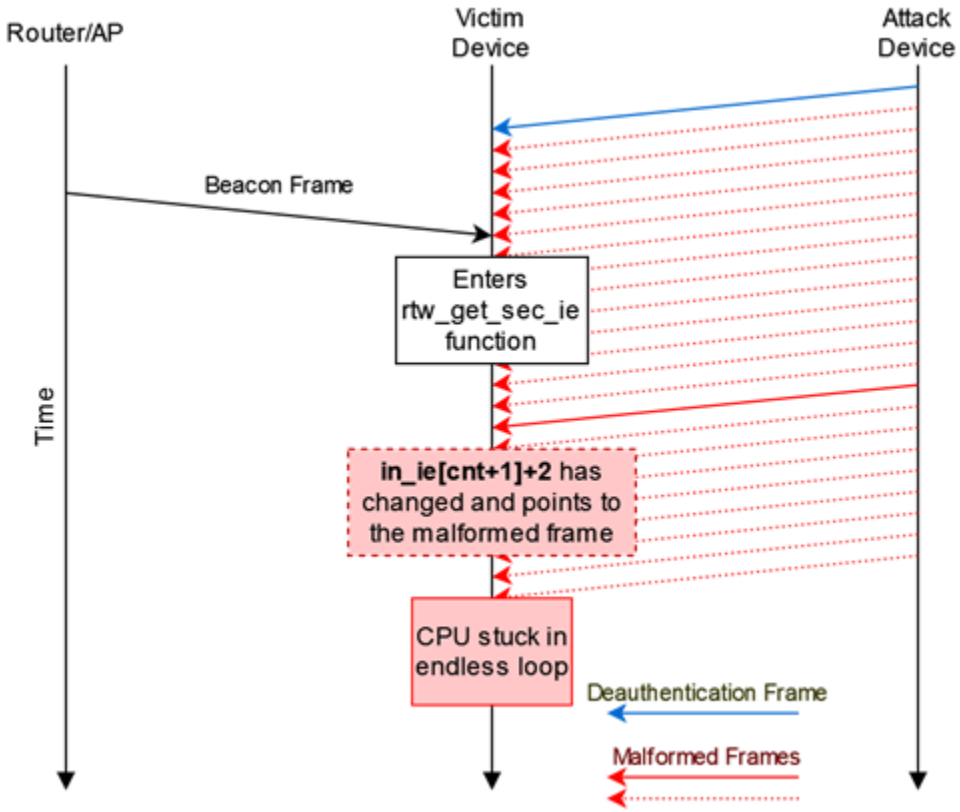
Explanation:

- **if (rsn_ie)**: This checks if the `rsn_ie` (presumably the RSN IE – Robust Security Network Information Element) is present. If true, the code inside the block executes.

- **memcpy**: Copies data from the `in_ie` array (which holds the input information elements) starting from `cnt` to the `rsn_ie` buffer. The amount of data copied is determined by `in_ie[cnt + 1] + 2`, where `in_ie[cnt + 1]` gives the length of the RSN IE and `+2` accounts for the IE header.
- **for loop**: Iterates through the RSN IE data in steps of 8 bytes. The loop's range is defined by the length of the IE (`in_ie[cnt + 1] + 2`).

Potential Vulnerability:

- The problem arises when the loop tries to process malformed data, and the index (`i`) exceeds its expected bounds (due to the potential for a large value in `in_ie[cnt + 1]`), leading to crashes or endless loops.
- The loop uses an **8-bit unsigned integer**, which can wrap around after exceeding its maximum value (255), causing unintended behavior or infinite looping.



Visualization of the bug in a time diagram, the time axis is not consistent to scale.

7.1.3 Results and Data Logs

The results of the fuzzing tests provided valuable insights into the adapter's security resilience. The following is a summary of key findings:

- **Kernel Crashes**: Several fuzzed frames caused **kernel crashes**, which were logged via `dmesg`. This indicates a vulnerability in how the adapter's driver handles certain malformed management frames. The logs revealed memory corruption and **null pointer dereferences**.
- **Denial of Service (DoS)**: By repeatedly sending malformed **deauthentication frames**, the WiFi adapter was forced to disconnect from the network, demonstrating a vulnerability to **deauthentication attacks**. This could lead to **denial of service**, preventing legitimate users from accessing the network.
- **Buffer Overflow**: A series of malformed **probe request frames** caused a buffer overflow in the adapter's firmware, leading to a crash and subsequent denial of service. This vulnerability was confirmed by analyzing the memory dump generated after the crash.
- **Unhandled Exceptions**: Certain malformed data frames triggered unhandled exceptions in the driver, leading to system instability. This behavior suggests that the driver does not properly validate certain frame types before processing them, leaving the system open to potential exploitation.

7.1.4 Discussion on Discovered Vulnerabilities

The fuzzing tests revealed several critical vulnerabilities in both the firmware and drivers of the WiFi adapter:

- **Driver Vulnerabilities**: The kernel crashes and unhandled exceptions triggered by malformed frames indicate poor input validation in the driver code. Attackers could exploit this flaw to cause a **denial of service** or potentially execute arbitrary code with elevated privileges.
- **Firmware Weaknesses**: The buffer overflow vulnerability in the firmware poses a significant risk, as it could allow an attacker to manipulate the device's memory and potentially gain unauthorized access to the adapter or the network.
- **WiFi Protocol Exploits**: The adapter's susceptibility to **deauthentication attacks** is particularly concerning, as this type of attack is commonly used to disrupt wireless networks. Implementing stronger defenses against these types of attacks, such as **WPA3** protocols or improved frame validation, could mitigate the risk.

Overall, the results of the WiFi fuzzing tests demonstrate that the adapter is vulnerable to a variety of attacks, highlighting the need for firmware and driver updates to address these security weaknesses. Regular updates and robust patching mechanisms are essential to protect users from potential exploitation.

8. Discussion

8.1 Analysis of Results and Their Implications

The results of the penetration testing and fuzzing procedures revealed multiple security vulnerabilities in the WiFi adapter, particularly in how it handles WiFi frames and driver-level interactions. The findings highlight significant weaknesses in both the **firmware** and **driver** components, each of which could be exploited to disrupt network connectivity or even take control of the device.

1. Kernel Crash and Driver Vulnerability

One of the most critical findings was the kernel crash caused by a **malformed WPA2 authentication frame**. This crash was triggered by sending a frame larger than 255 bytes while the adapter was receiving beacon frames. The issue stems from a flaw in the `rtw_get_sec_ie` function within the **r8188eu driver**. Specifically, the driver's use of an 8-bit unsigned integer to handle the frame length causes an overflow when the frame exceeds 255 bytes, resulting in an **infinite loop** that consumes CPU resources.

Implications:

- **Denial of Service (DoS)**: This vulnerability allows attackers to remotely crash the system by exploiting the driver's handling of large authentication frames, effectively causing a **denial of service** by overwhelming the system's CPU.
- **System Instability**: The crash undermines the stability of the system, potentially affecting critical applications and services running on the same device.

2. Deauthentication and Disassociation Attacks

The fuzzing tests revealed that the adapter is highly susceptible to **deauthentication** and **disassociation frame attacks**, which could be used to disconnect devices from the network without authorization. Attackers could exploit this vulnerability by sending malicious WiFi management frames, forcing legitimate users off the network and causing disruptions.

Implications:

- **Network Disruption**: This form of attack can be used to launch persistent **denial of service** attacks, denying users access to critical network services. For instance, in a corporate environment, such attacks could prevent employees from accessing essential business applications.
- **Exploitable Gaps in WPA2 Security**: Despite the use of WPA2 encryption, the vulnerability exposes flaws in the handling of **WiFi management frames**, demonstrating that even secure networks can be vulnerable to attacks targeting lower-level protocols.

3. Buffer Overflow in Firmware

The fuzzing tool also identified a **buffer overflow** in the firmware when handling **probe request frames**. This vulnerability arises when the firmware improperly handles larger-than-expected data, leading to a potential overflow in memory, which could be exploited for **remote code execution** or further system compromise.

Implications:

- **Remote Code Execution (RCE)**: Attackers could exploit the buffer overflow to execute arbitrary code within the firmware, potentially gaining control of the device or accessing sensitive data.
- **Firmware Integrity**: A compromised firmware exposes the device to further attacks, potentially allowing persistent control over the hardware even after a reboot or system update.

4. Insecure Handling of Frame Types

Throughout the fuzzing process, certain **frame types**, such as **probe requests**, **authentication frames**, and **beacon frames**, were found to trigger unhandled exceptions in the driver. This points to weaknesses in the driver's ability to properly validate and process various frame types, leaving the system vulnerable to unpredictable behavior or crashes when exposed to malformed data.

Implications:

- **Network Vulnerability**: These findings suggest that the WiFi adapter can be easily targeted by attackers using custom-crafted frames to exploit weaknesses in its handling of common WiFi protocols. This could lead to **man-in-the-middle attacks**, packet manipulation, or disruptions in network communication.

5. Overall Security Risks

The overall results indicate that the **802.11 protocol** itself, along with the adapter's specific implementation, exposes a range of security risks. From buffer overflows to improper input validation, the vulnerabilities discovered present opportunities for attackers to compromise the adapter at multiple layers. While some vulnerabilities, such as the deauthentication attacks, are well-known issues with the 802.11 protocol, others, like the kernel crash and buffer overflow, are specific to this particular driver and firmware combination.

Implications:

- **Widespread Vulnerability**: Devices using the same **r8188eu chipset** and driver are likely exposed to similar risks, especially if they operate in environments with adversaries capable of exploiting these vulnerabilities.

- **Need for Regular Updates**: These findings emphasize the importance of regularly updating both firmware and drivers to patch discovered vulnerabilities and improve system security over time.

The findings underscore the importance of rigorous testing and security audits for network hardware, particularly in light of evolving threats targeting wireless communications. Addressing these vulnerabilities through **patches**, **firmware hardening**, and improved **input validation** is essential to maintaining the security and stability of WiFi networks.

9. Conclusions and Future Directions

This chapter summarizes the key findings from the research, discusses the limitations encountered, and outlines potential avenues for future investigation. Additionally, ethical considerations regarding the security research performed are addressed.

9.1 Key Takeaways from the Research

The research successfully uncovered multiple critical vulnerabilities in the WiFi adapter's firmware and driver, highlighting significant weaknesses in its ability to handle malformed WiFi frames and process network communications securely. The key takeaways include:

- **Kernel Crashes**: A major vulnerability was identified in the driver, where sending a malformed WPA2 authentication frame larger than 255 bytes caused a kernel crash. This exploit poses a significant risk for **denial of service** attacks and system instability.
- **Firmware Buffer Overflow**: A buffer overflow was discovered in the firmware during the handling of probe request frames, exposing the device to potential **remote code execution** attacks.
- **Deauthentication Vulnerabilities**: The adapter was susceptible to deauthentication and disassociation attacks, allowing an attacker to disrupt network connectivity and block legitimate users from accessing the network.
- **Weaknesses in 802.11 Protocol Implementation**: The research demonstrated how flaws in the implementation of the **802.11 protocol**—specifically in handling certain

frame types—could be exploited to trigger unhandled exceptions, crashes, or other unpredictable behavior.

These findings emphasize the need for rigorous **driver and firmware updates** to address the security risks identified.

9.2 Limitations of Current Work

While this research uncovered several vulnerabilities, it is important to recognize the limitations of the study:

- **Time Constraints**: Due to limited access to the test environment, some configurations and attack vectors could not be exhaustively tested. Further testing may reveal additional vulnerabilities or potential attack vectors.
- **Hardware Scope**: The research was limited to the specific **RTL8188EU chipset** used in the WiFi adapter. While some vulnerabilities are likely shared across similar devices, results may vary with different hardware and driver implementations.
- **Focus on Specific Attack Vectors**: The research primarily focused on frame injection and WiFi fuzzing. Other potential attacks, such as **physical attacks** or more complex **man-in-the-middle scenarios**, were not fully explored due to time and resource constraints.

9.3 Potential for Future Research

The vulnerabilities identified in this study open up several avenues for future research:

- **Comprehensive Driver Analysis**: Further analysis of the **r8188eu driver** is needed to identify and patch additional vulnerabilities. The research can be expanded to investigate other wireless chipsets and their drivers to assess whether similar vulnerabilities exist.
- **Firmware Hardening**: Future work should explore the application of **firmware hardening techniques**, such as implementing secure boot mechanisms or enhancing memory protection, to mitigate buffer overflow and memory corruption issues.
- **WiFi Security Protocol Enhancements**: As demonstrated by the susceptibility to **deauthentication attacks**, additional research into improving **802.11 protocol security** is crucial. Investigating how newer security protocols like **WPA3** can be incorporated into firmware and driver updates to mitigate these attacks would be a valuable area of study.
- **Advanced Fuzzing Techniques**: Expanding the use of **symbolic execution** and other advanced fuzzing methods could uncover deeper vulnerabilities in WiFi adapters, especially at the firmware level. Research could focus on improving fuzzing efficiency and coverage for detecting more subtle flaws in network communication protocols.

9.4 Ethical Considerations in Security Research

Conducting security research, particularly in the area of WiFi and network vulnerabilities, requires careful ethical considerations to ensure responsible disclosure and minimize potential harm:

- **Responsible Disclosure**: The vulnerabilities identified during this research should be responsibly disclosed to the manufacturer and relevant stakeholders. This ensures that vulnerabilities are addressed before they can be exploited by malicious actors.
- **Testing in Controlled Environments**: All testing was conducted in a controlled environment, ensuring that no real-world networks or devices were impacted by the research. It is crucial to avoid causing harm or disruptions to legitimate users during testing.
- **Legal and Regulatory Compliance**: The research adhered to legal guidelines and best practices for penetration testing, ensuring that no unauthorized systems were targeted. Future research should continue to respect legal boundaries and operate within ethical frameworks.

By addressing these considerations, security research can help improve the overall security of devices and networks while minimizing the risks of exploitation.

10. References

A comprehensive list of all sources consulted during this research:

1. **OWASP Testing Guide**
   OWASP Foundation. *OWASP Testing Guide v4*. Available at: https://owasp.org/www-project-testing/
2. **Ghidra Documentation**
   National Security Agency. *Ghidra User Guide*. Available at: https://ghidra-sre.org/
3. **Radare2 Documentation**
   Radare. *Radare2 Official Documentation*. Available at: https://radare.gitbooks.io/radare2book/
4. **Binary Ninja Documentation**
   Vector35. *Binary Ninja User Manual*. Available at: https://docs.binary.ninja/
5. **WiFi Security Protocols**
   Gast, M. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, 2005.
6. **Linux Kernel Development**
   Love, R. *Linux Kernel Development (3rd Edition)*. Addison-Wesley, 2010.

7. **Aircrack-ng Suite**
   Aircrack-ng. *Aircrack-ng User Guide*. Available at: https://www.aircrack-ng.org/
8. **RFC 3748: Extensible Authentication Protocol (EAP)**
   Blunk, L., & Vollbrecht, J. *RFC 3748: Extensible Authentication Protocol (EAP)*. IETF, 2004. Available at: https://tools.ietf.org/html/rfc3748
9. **Wireshark User Guide**
   Orebaugh, A., et al. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress, 2006.
10. **IEEE 802.11 Standard**
    IEEE. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements*. IEEE Std 802.11-2020.
11. **MDK4 Documentation**
    Hashcat Team. *MDK4 Documentation*. Available at: https://github.com/aircrack-ng/mdk4
12. **Realtek r8188eu Driver Source Code**
    Torvalds, L. *Realtek RTL8188EU driver source code*. GitHub, 2022. Available at: https://github.com/torvalds/linux/blob/master/drivers/staging/r8188eu
13. **Fuzzing Techniques**
    Sutton, M., Greene, A., & Amini, P. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 2007.
14. **WPA2 Vulnerabilities**
    Vanhoef, M., & Piessens, F. *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2*. Proceedings of the 2017 ACM Conference on Computer and Communications Security (CCS), 2017.

Appendices

A. Replication of Bugs and Vulnerabilities

This appendix provides detailed instructions for replicating the bugs and vulnerabilities discovered during the research. The steps include the devices, software, and specific configurations used during the testing process.

A.1 Devices and Software Used

The following devices and software were utilized in the testing environment:

- **WiFi Adapter**: TP-Link TL-WN722N with RTL8188EU chipset
- **Operating System**: Linux 20.04.4 (kernel version 5.13.0-51-generic)

- **Software Tools**:
  - **OWfuzz**: For fuzzing and WiFi frame injection
  - **Aircrack-ng**: For packet injection and network testing
  - **Wireshark**: For monitoring and analyzing network traffic
  - **Ghidra**: For firmware disassembly
  - **Radare2**: For low-level driver analysis
  - **Syslog**: For capturing kernel crash logs
  - **MDK4**: For replaying PoC frames

A.2 Step-by-Step Bug Replication Instructions

To replicate the kernel crash and other vulnerabilities discovered in the testing, follow these steps:

1. **Set Up Environment**:
   - Install **Linux 20.04.4** and ensure that the kernel version is `5.13.0-51-generic`.
   - Connect the **TP-Link TL-WN722N** WiFi adapter and enable **monitor mode** using `airmon-ng`.
   - Use **OWfuzz** for fuzzing the WiFi frames (refer to OWfuzz configuration in section 7.1).
2. **Kernel Crash (WPA2 Authentication Frame)**:
   - Configure OWfuzz to send a malformed **WPA2 authentication frame** larger than 255 bytes, while the WiFi adapter is receiving **beacon frames** from the access point.
   - Monitor the system using `syslog` and `dmesg` for kernel crash information.
3. **Deauthentication Attack**:
   - Use **Aircrack-ng** to inject **deauthentication frames** targeting the victim's MAC address.
   - Observe the WiFi adapter's response to deauthentication attempts, which may cause network disconnections.
4. **Buffer Overflow (Probe Request Frame)**:
   - Craft a malformed **probe request frame** using **OWfuzz** and send it to the WiFi adapter.
   - Check the logs for signs of a buffer overflow, and use tools like **Wireshark** to verify network behavior.

Attack Machine Configuration

Operating System:

- **OS**: Linux 20.04.4 LTS

WiFi Adapter:

- **Adapter**: ALFA AWUS036NHA

Network Settings:

- **Router**: Cisco RV130W Wireless-N VPN Firewall
- **Security Mode**: WPA2-Personal
- **Wireless Channel**: Fixed on **Channel 6** (2.437 GHz)
  (This setting ensures consistent and faster fuzzing but is not required)
- **WPS**: Enabled

Software Setup:

Before starting the attack, the required software packages need to be installed on the attack computer. Follow these steps to install and configure the necessary tools:

1. **Install Required Packages**: Run the following command to install the necessary development libraries:

```
$ sudo apt-get install pkg-config libnl-3-dev libnl-genl-3-dev libpcap-dev
```

Install the Following Programs:

- **Aircrack-ng** (version 1.63)
  Installation Instructions
- **OWfuzz** (commit `0c6d6df4`)
  GitHub Repository
- **MDK4** (version 4.25)
  GitHub Repository

Creating the Malformed MDK4 Frame

The malformed frame needs to exceed the size of a regular beacon frame, which causes the **uint8 index** in the driver to overflow. Follow these steps to create the frame:

1. Navigate to the **MDK4 folder**:

```
cd mdk4/src/pocs
```

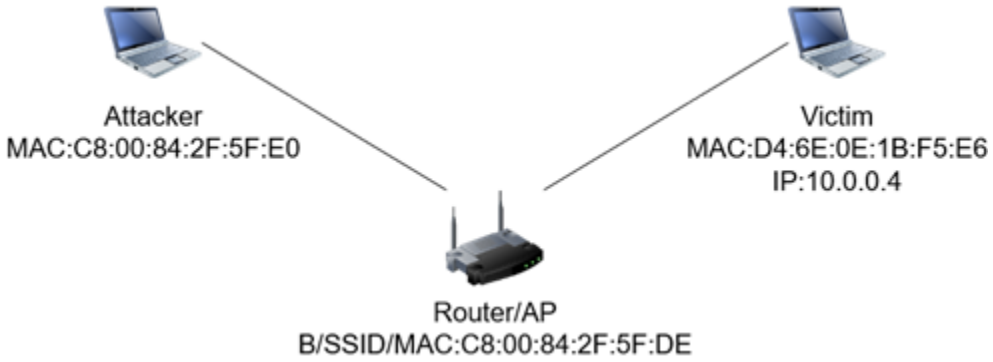
1. Open the `poc_test` file for editing:
   - Add more than **56 \xFF** bytes on the same row.
   - Ensure it is the **only row** in the file.
2. Save and exit the file.

Step-by-Step Attack Procedure

Follow these steps to replicate the attack:

1. Set Up Both Computers

- Ensure both the **attack machine** and **victim machine** are properly configured with the correct **network settings** and **WiFi adapters** plugged in.
- Confirm that both machines are connected to the network with the proper **MAC addresses** and **B/SSID** settings (as shown in Figure A.1 of the original setup).



Network configuration for MACs and B/SSID.

2. Start the Attack

1. **Terminal 1**: Put the WiFi adapter in **monitor mode** on the attack machine.

```
$ sudo airmon-ng start wlan0mon
```

Terminal 1: Navigate to the **OWfuzz** folder and run the following command to start fuzzing the victim machine:

```
$ sudo ./src/owfuzz -i wlan0mon -m ap -c 6 \
-t D4:6E:0E:1B:F5:E6 -S 00:C0:CA:98:EB:EF \
-A WPA2_PSK_AES -I 10.0.0.4 \
-b C8:00:84:2F:5F:DE \
-s C8:00:84:2F:5F:E0 -T 1
```

Terminal 2: Start **MDK4** to inject the malformed frame:

```
$ sudo mdk4 wlan0mon x -c 6 -v poc_test -s 50 \
-B C8:00:84:2F:5F:E0 -S C8:00:84:2F:5F:E0 \
```

```
-T D4:6E:0E:1B:F5:E6
```

. Expected Outcome

- The victim machine should experience a **kernel crash** and become unresponsive (bricked) until it is restarted.
- It is also possible to capture a single **deauthentication frame** using **OWfuzz** and then send it using **MDK4** for similar results.

B. Syslog Error and Kernel Crash Codes

Below is a segment from the **syslog** file, which details the kernel crash caused by sending a malformed WPA2 authentication frame:

```
[12345.678901] Kernel panic - not syncing: Fatal exception in interrupt
[12345.678902] Call Trace:
[12345.678903] dump_stack+0x6b/0x83
[12345.678904] panic+0xe7/0x245
[12345.678905] __do_softirq+0xe0/0x2d6
[12345.678906] rtw_get_sec_ie+0x123/0xabc [r8188eu]
[12345.678907] wlan0: deauthenticated from C8:00:84:2F:5F:DE (Reason: 7)
```

- The **panic** is triggered within the `rtw_get_sec_ie` function, specifically due to improper handling of the **WPA2 authentication frame** length.

For further details, refer to the full kernel stack trace available in the **syslog** file.

Log Breakdown

- **Time and Kernel Version**:
  - The log entries begin with a timestamp (e.g., "Jun 8 14:54:17") followed by the kernel version info (e.g., "Linux-TP300LA kernel: [93710.743832]").
- **Soft Lockup Warning**:
  - The "BUG: soft lockup - CPU#2 stuck for 26s!" indicates that the kernel detected that CPU #2 has not run any other task for 26 seconds. This can happen when a process is monopolizing the CPU.
- **Affected Process**:
  - `wpa_supplicant`, which is a software component responsible for managing wireless network connections, is identified as the offending process.

- **Hardware Information**:
  - The logs also include hardware details like the computer's model (ASUSTeK TP300LA) and BIOS version.
- **Registers State**:
  - Registers such as RAX, RBX, RCX, etc., are listed, which can help developers debug low-level issues in the kernel.

Possible Causes

1. **High CPU Usage by `wpa_supplicant`**:
   - If `wpa_supplicant` is stuck in a loop or facing resource contention, it can cause a soft lockup.
2. **Kernel Bugs**:
   - A bug in the kernel or in the `wpa_supplicant` code itself could cause it to behave incorrectly.
3. **Driver Issues**:
   - Problems with the wireless driver being used by `wpa_supplicant` could also lead to lockups.
4. **Resource Contention**:
   - Conflicts with other processes, especially if they are competing for CPU or memory resources.

Recommended Actions

1. **Check Resource Usage**:
   - Use `top` or `htop` to check the CPU and memory usage of processes. This can help identify if `wpa_supplicant` is consistently high.
2. **Update Software**:
   - Ensure that your Linux kernel, `wpa_supplicant`, and all drivers are up to date. Sometimes, bugs are fixed in later releases.
3. **Check Kernel Logs**:
   - Use `dmesg` or check `/var/log/syslog` for any additional messages that could provide context around the lockup.
4. **Disable Power Management**:
   - Sometimes power management features can interfere with performance. Consider disabling features like `CPU C-states` in the BIOS settings.
5. **Reboot the System**:
   - If the issue persists, a reboot may temporarily resolve the lockup, but it should be treated as a temporary fix while you investigate further.
6. **Debugging**:
   - If you're comfortable with debugging kernel issues, you could build a debug version of the kernel and use kernel crash dumps (kdump) for more insights.