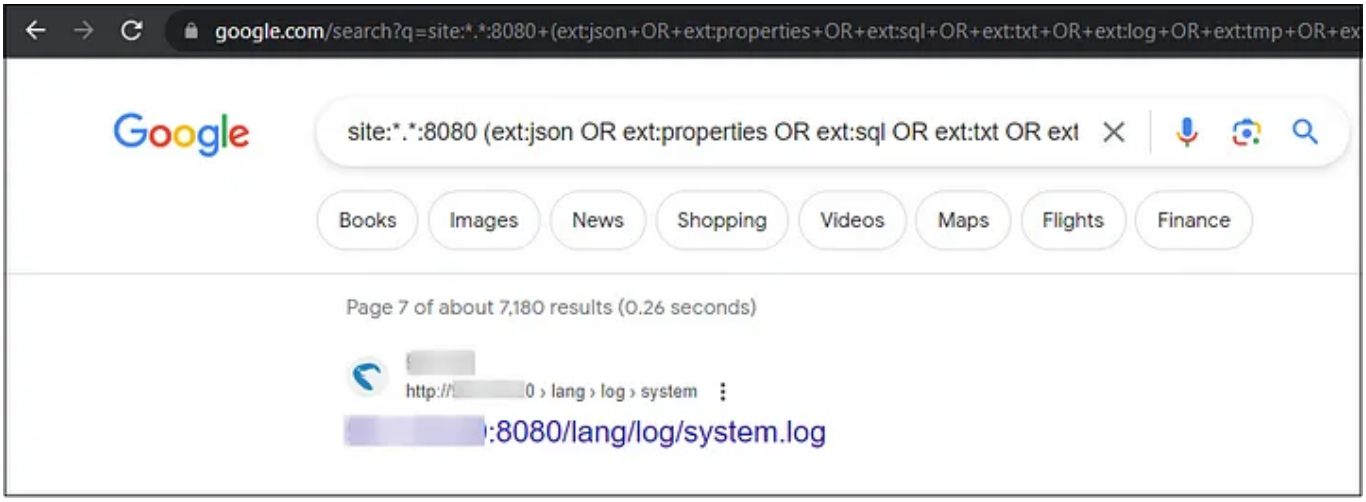
Inside the Router: How I Accessed Industrial Routers and Reported the Flaws

> *without any negative impact. Therefore, the following vulnerability content is for discussion and research purposes only.*

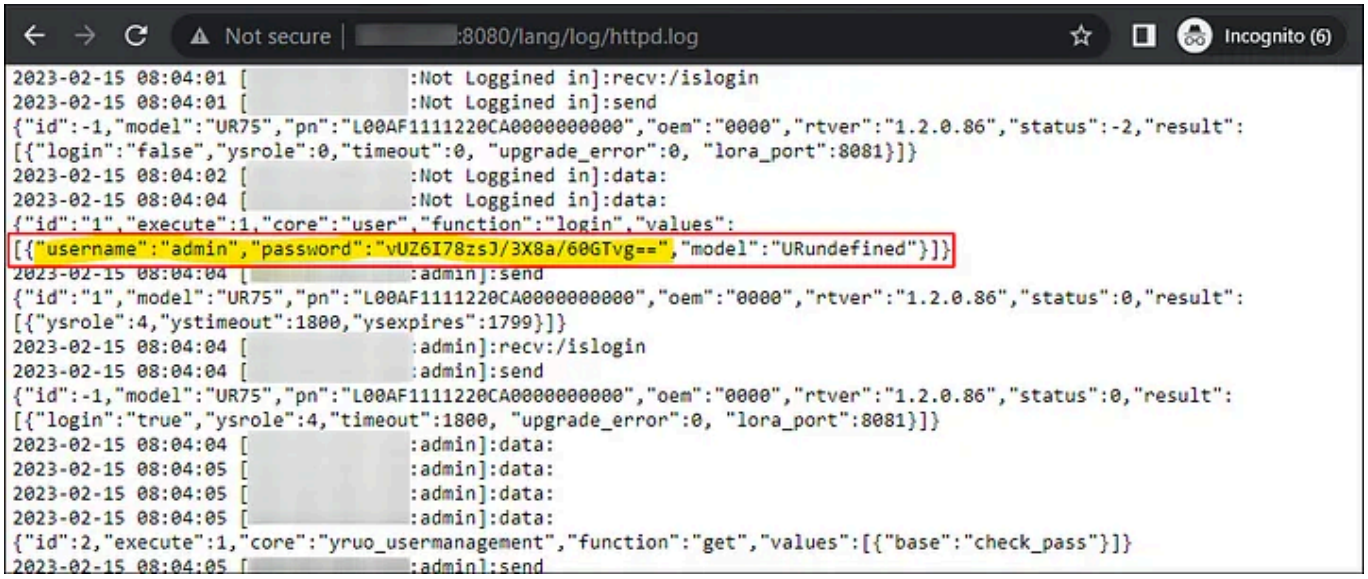
CHAPTER 1

Accidental Discovery

A few months back, I shared a search trick on GitHub that allowed you to find thousands of leaked keys and secrets from public repositories. You can check out that search syntax right here. As I was experimenting with it, I



As a result, I came across the website



From a quick Google search, I found out that Ursalink is a manufacturer of IoT products in the industrial sector. It was a vendor of remote monitoring, data collection, and automation devices for use in various industrial applications.

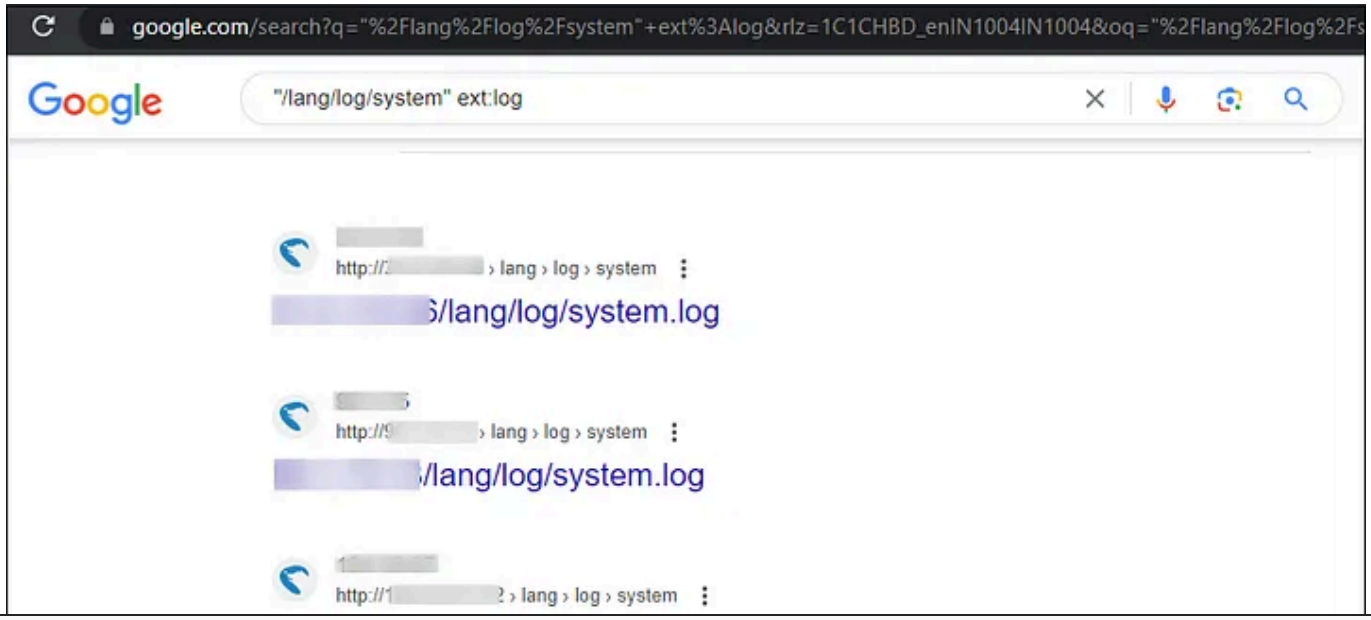
I guessed it might be a router login. Upon closer examination of the login page, I came to know that it utilized a JavaScript file called `login.js`.

To further investigate, I quickly created a decryption code and successfully decrypted the previously copied password. Here is the decryption code I used:

```
console.log(CryptoJS.AES.decrypt("vUZ6I78zsJ/3X8a/60GTvg==",
  CryptoJS.enc.Utf8.parse("1111111111111111"), {
      iv: CryptoJS.enc.Utf8.parse("2222222222222222"),
      mode: CryptoJS.mode.CBC,
      padding: CryptoJS.pad.Pkcs7
  }).toString(CryptoJS.enc.Utf8));
```

UR75 V1, a first-generation Industrial Cellular Router model

cellular router. Therefore, it is quite difficult to get SIM-owner/router-owner information to inform/notify about this issue.



Medium

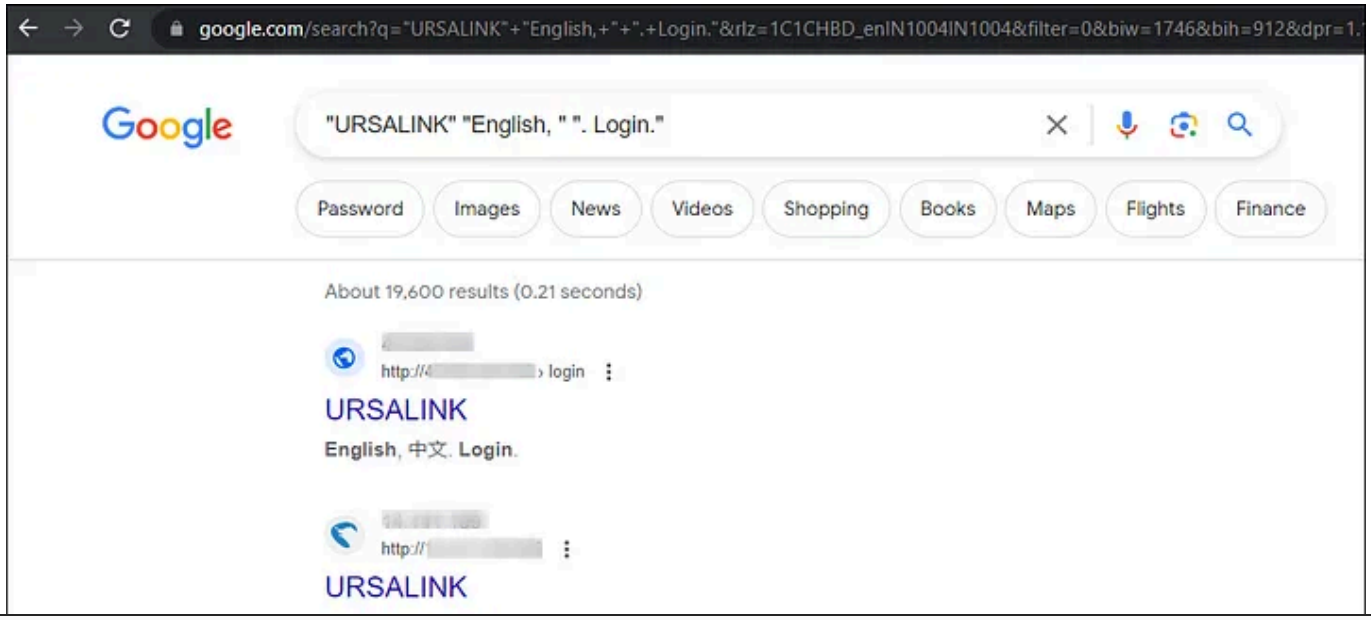
Sign up to discover human stories that deepen your understanding of the world.

Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

✦ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app



Medium

Sign up to discover human stories that deepen your understanding of the world.

Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

✦ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

To demonstrate the impact, and verify the vulnerability at scale, I created a Python script to test all of those results, and you can find it in my GitHub repository.

The script allows the testing of a router's console URL or a list of URLs from a text file and quickly retrieves the admin password. Passed the result of the vulnerable list of URLs to my script and observed cleartext admin credentials.

As a result — most routers were vulnerable, and I collected administrator credentials. I even logged in to a few devices as proof. I identified that the Ursalink Industrial Cellular Router series UR5X, UR32L, UR32, UR35, and UR41 were vulnerable and others may also be vulnerable.

This vulnerability becomes even more severe as **some routers allow the sending and receiving of SMS messages.** An attacker could exploit this functionality for fraudulent activities, potentially causing financial harm to the router owner.

Additionally, I stressed the need to **avoid the practice of hardcoding keys or IVs** within the application code. While I acknowledged that the encryption layer was intended to provide an extra layer of security on top of TLS, I pointed out that its current implementation might not be as effective as intended. **This encryption could be easily cracked and introduce unnecessary overhead to both the browser and the application, affecting overall performance.** Therefore, I suggested reconsidering the necessity of this layer or exploring alternative, more secure algorithm implementations.

In response to my report, the company thanked me for the detailed report

So, I received a firmware file named **35.3.0.7.bin**. Using the `file 35.3.0.7.bin` command, I found out that it is a zip archive. Running `binwalk 35.3.0.7.bin`, I found that this file was encrypted/password-protected and it contained two compressed files: router.tar and upgrade_tool.tar.gz.

1. **`router.tar`** was the big deal here; it held all the crucial firmware components like the kernel and file system.
2. On the other hand, **`upgrade_tool.tar.gz`** wasn't as important. It contained some scripts to extract the firmware data from `router.tar` and perform a

allow program execution within the filesystem, I opted not to proceed with it.

I didn't have access to the physical device (which happened to be an Industrial Cellular Router). So, **I decided to attempt emulating the provided firmware on my Debian Linux (Ubuntu) machine using a tool called QEMU (Quick Emulator)**, the generic open-source emulator and virtualizer.

Now, emulating router firmware directly on a Linux Debian system isn't exactly a walk in the park. You see, router firmware is designed for embedded systems with different CPU architectures (like ARM or MIPS) than what a typical Linux Debian system uses (x86 or x86_64). After reading the router's specifications, I found out that this particular router used a **32-bit**

Emulating one architecture on another can be difficult and requires tools like QEMU, which can be a bit of a hassle to set up.

So, I rolled up my sleeves and installed QEMU for ARM with the command `sudo apt install qemu-system-arm`. You can choose between `qemu-system-arm` or `qemu-system-aarch64` to simulate a 32-bit ARM machine. **QEMU's ARM system emulation requires you to specify a board model using the `-M` or `-machine` option.** This is where `ur35.dtb` file comes into play, the file I extracted earlier from the router.

A quick look at `ur35.dts` showed that the model was “**Freescale i.MX6 UltraLite 14x14 EVK Board**”.

We're getting closer!

With everything set up, my ultimate goal was to gain access to the root shell and the router's administration web panel. This would allow me to analyze how it handled user input (debugging) and find potential vulnerabilities.

Putting all the pieces together, I came up with the following command:

```
qemu-system-arm -M mcimx6ul-evk -kernel zImage_signed.bin -initrd
```

5. `-append “root=/dev/ram0 init=init”`: This option provides a kernel command line that will be passed to the kernel during boot. It specifies two boot parameters: **1.** `root=/dev/ram0` — This sets the root filesystem to be loaded from RAM (/dev/ram0) initially. **2.** `init=init` — It instructs the kernel to execute the traditional init process as the initial user-space program during boot
6. `-dtb ur35.dtb`: This option specifies the device tree binary (DTB) file to be used. Device tree files describe the hardware configuration to the kernel, and “ur35.dtb” is the one specified here.

I extracted the `luci2-io` binary file from the firmware and started analyzing it using **Ghidra**.

Lastly, I want to extend my heartfelt thanks to **Milesight** for their proper coordination and for providing the firmware.

Disclosure Timeline

- **June 22, 2023:** Initial notification to the vendor requesting assistance in obtaining the appropriate email address for reporting the security issue.
- **June 26, 2023:** Response received from Kevin Huang, Senior Technical Specialist, instructing me to share the vulnerability details via email.

Thank you for reading.

Keep learning, and stay safe and healthy! 😇

. . .

Who am I?

To briefly introduce myself, my name is Bipin Jitiya and I am the founder of Cuberk Solutions.

Written by Bipin Jitiya

Follow

1.5K Followers

Security Enthusiast | Professional Penetration Tester | Web & Mobile Application Developer | Reverse Engineer | Learn more at https://win3zz.com/