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

Router Vulnerability Hunt, From Google Dorks to Firmware Emulation — The Full Story

Bipin Jitiya · Follow

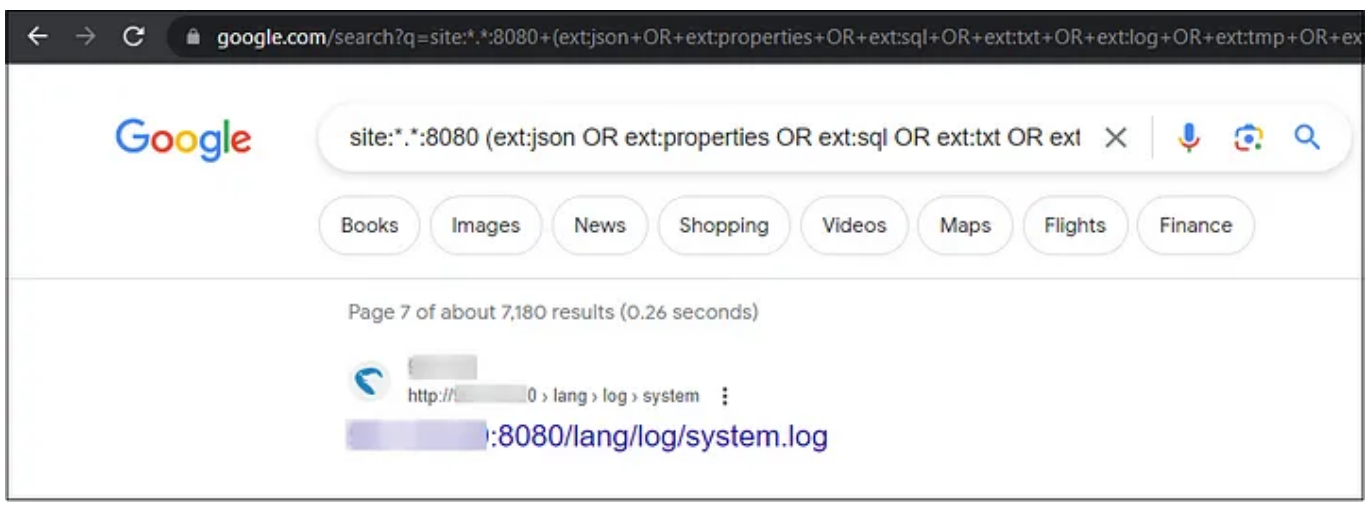
13 min read · Oct 1, 2023

Hello, World! ❤️‍🩹

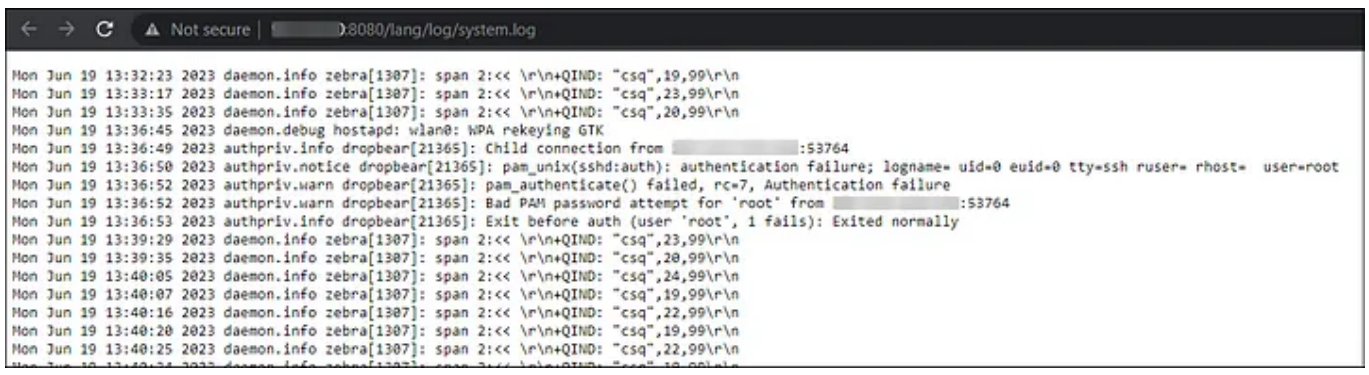
Today, I have an exciting story about how I exposed admin passwords and gained access to thousands of **3G/4G/5G Industrial Cellular Routers** with the help of some old-school vulnerabilities (or rather, misconfigurations).

> *Before proceeding please note that the **following vulnerabilities have been immediately identified and fixed.** The manufacturer proactively communicated the vulnerability situation and promptly updated the software to address the vulnerability risks. **I confirm that this issue has been resolved by August 2023 without any negative impact. Therefore, the following vulnerability content is for discussion and research purposes only.***

Pretty simple, right? This small piece of code will help me find sensitive files lik
ex



As a result, I came across the website http://[REDACTED]:8080/lang/log/system.log. Upon visiting this website, I discovered that it was **publicly disclosing system logs, which included internal information.**



By deleting `system.log` from the URL and navigating back to one directory, I noticed **a server misconfiguration that allowed me to view the contents or list of files within that particular directory.** In this directory, I found an

1. ../
   modified: Mon, 19 Oct 2020 15:26:00 GMT
   directory - 0.00 kbyte

2. **Default.bin**
   modified: Fri, 28 Feb 2020 03:45:45 GMT
   application/octet-stream - 11.62 kbyte

3. **httpd.log**
   modified: Tue, 20 Jun 2023 08:33:37 GMT
   text/plain - 264.99 kbyte

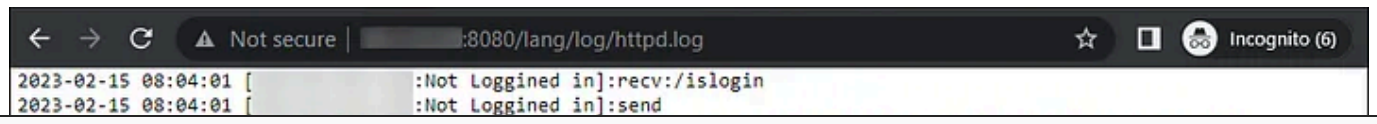
4. **httpd.log.old**
   modified: Mon, 28 Nov 2022 20:40:15 GMT
   application/octet-stream - 1019.42 kbyte

5. **sd_detect.log**
   modified: Thu, 30 Mar 2023 23:48:41 GMT
   text/plain - 0.00 kbyte

6. **system.log**
   modified: Tue, 20 Jun 2023 08:33:27 GMT
   text/plain - 375.35 kbyte

7. **system.log.old**
   modified: Mon, 19 Jun 2023 20:32:08 GMT
   application/octet-stream - 2048.02 kbyte

Upon reviewing its contents, I identified that **usernames and encrypted passwords were being logged.** 😳



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`.

Upon analyzing the JavaScript code, I determined that the **application was using the AES algorithm in CBC Cipher mode, with a hardcoded secret key and initialization vector (IV).** Here is a formatted version of the same JavaScript code snippet:

```
$("#login").click(function() {
    var e = $("#username").val(),
        n = $("#password").val();
    if (0 == e.length) return void $(".error").html(language_class.login.error.usern
    if (0 == n.length) return void $(".error").html(language_class.login.error.passw
    var o = CryptoJS.enc.Utf8.parse("1111111111111111"),
        t = CryptoJS.enc.Utf8.parse("2222222222222222"),
        l = CryptoJS.enc.Utf8.parse(n),
        i = CryptoJS.AES.encrypt(l, o, {
            iv: t,
            mode: CryptoJS.mode.CBC,
            padding: CryptoJS.pad.Pkcs7
        });
    n = CryptoJS.enc.Base64.stringify(CryptoJS.enc.Hex.parse(i.ciphertext.toString()
        .toUpperCase()));
```

But would this password actually get me inside the application? To validate the severity of this vulnerability, I attempted to log into the application using the username and cleartext password.

UR75 V1, a first-generation Industrial Cellular Router model

Along with this, I also did port scanning using `nmap` and found 4 open ports including TCP port 22.

With the help of the same credentials, **I managed to log in to the router console using SSH.**

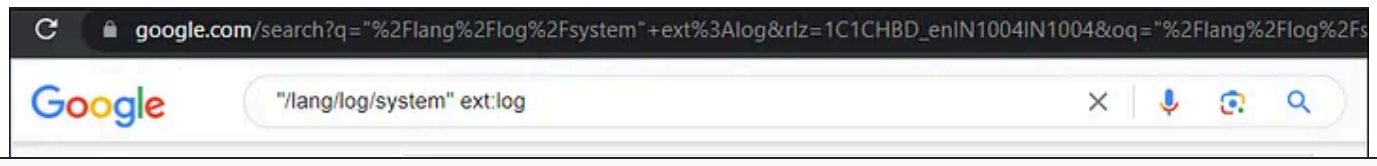
To find the owner of the router or the linked organization, I used bgp.he.net to [redacted], I de[redacted]a Canadian telecom company.

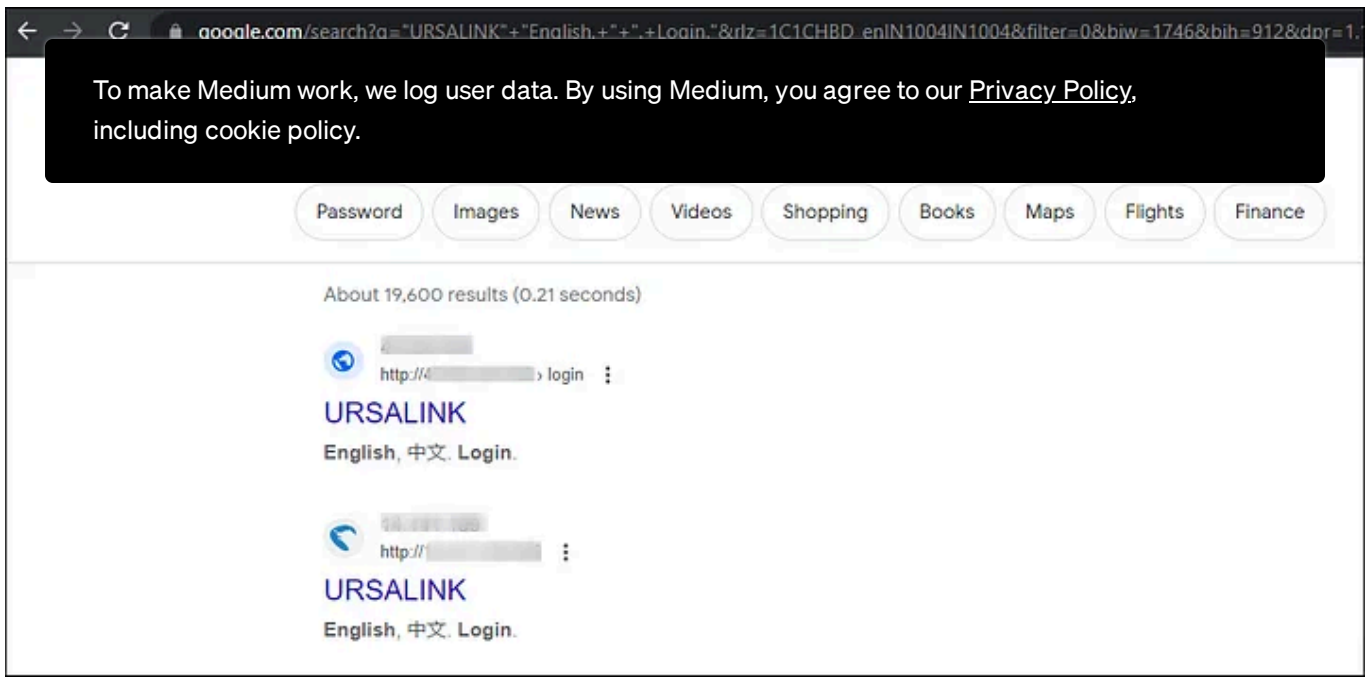
After a little analysis and gathering information from the internet, I concluded that the IP address is assigned by Telus to the SIM card inside the cellular router. Therefore, it is quite difficult to get SIM-owner/router-owner information to inform/notify about this issue.

. . .

But I wasn't done yet!

I wanted to see if other routers were also vulnerable. I queried again with the new Google Dorks `"/lang/log/system" ext:log`, `"URSALINK" "English" "Login"` and confirmed that a bunch of routers were also vulnerable.





Google Dork : `"URSALINK" "English" "Login"`

I was not satisfied with the limited results, so I used the Shodan Search Engine to get more such routers using the `http.html:rt_title` search query filter.

To demonstrate the impact, and verify the vulnerability at scale, I created a Py
re

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.

fu
th

After logging in, go to **General Settings** in the **System** menu, and access the **SMS** tab for messaging.

recommended the implementation of appropriate access controls to limit th

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 and confirmed that the vulnerability was a known issue. They assured me that the vulnerability had already been resolved/fixed in their latest firmware. They also provided the latest firmware version for verification purposes.

. . .

CHAPTER 2

The Art of Firmware Emulation

In this chapter, I won't be able to share the findings or other confidential details, but I'm excited to share some essential steps with you. If you've ever found yourself in a similar situation, dealing with firmware and emulation, this may be the insight you need.

I decrypted `35.3.0.7.bin` and extracted some important files, namely `filesystem.squashfs`, `ur35.dtb`, and `zImage_signed.bin`, all from the `router.tar`. But then, the question was, "**How do I use these files?**"

One approach considered was using `binwalk -Me filesystem.squashfs` to extract the contents of a Squashfs filesystem for analysis, but since it doesn't allow program execution within the filesystem, I opted not to proceed with it.

firmware on my Debian Linux (Ubuntu) machine using a tool called QEMU (Q

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 ARM architecture.**

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

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

This is what the **i.MX6UltraLite Evaluation Kit (EVK)** board looks like

Next, I needed to select a machine or board model for QEMU. I found it by running `qemu-system-arm -machine help | grep i.MX6`, which led me to choose the "**mcimx6ul-evk**" board.

Let's break down the command step by step:

1. with an ARM-based processor architecture.
2. `-M mcimx6ul-evk`: This specifies the machine or board model to emulate. In this case, it's set to emulate the "**mcimx6ul-evk**" board.
3. `-kernel zImage_signed.bin`: This option specifies the kernel image file to be loaded into the virtual machine. "**zImage_signed.bin**" is the kernel image that will be used.
4. `-initrd filesystem.squashfs`: This option specifies the initial ramdisk (initrd) image file to be loaded into the virtual machine. It contains an initial file system that can be used during the boot process. In this case, it's "**filesystem.squashfs**"
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.
7. `-nographic`: This option tells QEMU to operate without a graphical user interface (GUI). It's useful for text-based or headless operations.
8. `-no-reboot`: This option tells QEMU not to automatically restart the virtual machine if it shuts down. This can be helpful when debugging or

A snapshot version of LEDE "Reboot" was running on the system. LEDE (Linux Embedded Development Environment) merged with the OpenWrt project in early 2018. OpenWrt is an open-source project that provides a Linux-based operating system (OS) and firmware for embedded devices, particularly routers and network devices.

I was more interested in web admin. The web admin interface details can typically be found in a configuration file. I began to inspect the router's configuration file to find relevant details.

After a little digging, I found a configuration file for the **uHTTPd web server**, wh
sy

I was looking for the controller file like `file-export` that was handling HTTP requests.

I found those files in `/www/cgi-bin/` directory.

After some debugging and analysis, I found several vulnerabilities, but due to company confidentiality, I can't share them here.

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.

Th[illegible]

Ke[illegible]

. . .

Who am I?

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

We're an information security company, we provide cutting-edge information security solutions to critical businesses with the intention of intelligently securing their IT environment. We offer a variety of vulnerability assessment and penetration testing services to our clients. If you have a minute or two to learn more about us, you can visit us here at www.cuberk.com

Cybersecurity | Programming | Technology | Software Engineering | Hacking

95 3

Bipin Jitiya

Simple story of some complicated XSS on Facebook

Hello, World! ❤️

Jun 21, 2020 812 2

Bipin Jitiya

Remote Command Execution in a Bank Server

A detailed article on how I exploited Remote Command Execution (RCE) with the help of...

Nov 18, 2022 485 4

See all from Bipin Jitiya

Recommended from Medium