FORESCOUT RESEARCH | VEDERE LABS

FINITE STATE

ROUGH AROUND THE EDGES

The State of OT/IoT Routers in the Software Supply Chain

August 6, 2024

CONTENTS

1. Executive Summary

Cellular routers connect critical Operational Technology (OT) and Internet of Things (IoT) devices to the internet. Electrical substations. Oil and gas fields. Temporary healthcare facilities — and more. These connections allow remote monitoring and control, especially where wired networks are difficult to deploy.

At the end of 2023, we studied vulnerabilities in OT/IoT router vendor: Sierra:21. In that research, Forescout Research — Vedere Labs discovered open-source software components are a key vulnerability. Today, we have widened our research lens to understand the state of software components in OT/IoT network devices beyond one vendor. **Our goal**: To understand risk in the software supply chain from existing ("n-day") vulnerabilities in the latest router firmware.

Supply-chain vulnerabilities are hard to eradicate because firmware images frequently depend on outdated components for compatibility — allowing threat actors to target many devices with a single exploit. However, identifying the intricate components used in common models of a specific class of devices is difficult at scale. To help, we partnered with Finite State, a leading Software Bill of Materials (SBoM) vendor, to analyze firmware images from popular routers: Acksys, Digi, MDEX, Teltonika, and Unitronics.

Key Findings

- **OpenWrt is everywhere**
  Four out of the five firmware analyzed run operating systems derived from OpenWrt. These four firmware images use heavily modified versions of the base OS, either by mixing and matching individual component versions with a base version or developing their own in-house components.

- **Outdated software components abound**
  Among 25 common components, the average open-source component is:
  - 5 years and 6 months old
  - 4 years and 4 months behind the latest release
  - The latest critical open-source components, including kernel and OpenSSL, are **not** being used

- **Known vulnerabilities are prevalent**
  On average, these firmware images had 161 known vulnerabilities in their most common components:
  - 68 with a low or medium CVSS score,
  - **69 with a high score**
  - **24 with a critical score**.
  - **20 exploitable n-day vulnerabilities affecting the kernel**

- **Security features are lacking across firmware images**
  On average:
  - 41% of binaries use relocation read-only (RELRO)
  - 31% use stack canaries
  - 65% use Non-eXecutable bit (NX)
  - 75% use position independent executable/code (PIE)
  - 4% use hard-coded run-time library search paths (RPath)
  - 35% have debugging symbols

Overall, we found positive correlations between the age of components, the number of known vulnerabilities and binary hardening practices. As expected, firmware with newer components tends to have fewer vulnerabilities and better binary protections. We observed a **decline of default credentials** which reduces the likelihood of exploitation under normal circumstances. However, we also noticed many **issues with custom patching** by vendors that do not change component versions and create confusion about what is *actually* vulnerable.

2. Selecting and Analyzing OT/IoT Cellular Routers

OT/IoT routers are frequent targets for cybercriminal botnets, nation-state APTs, such as Volt Typhoon (multiple times) or Fancy Bear, and hacktivists. We maintain an Adversary Engagement Environment (AEE) to monitor these activities. Vedere Labs reports have highlighted threat actors using public proof-of-concept (PoC) exploits to hijack routers and living-off-the-land (LotL) techniques for malicious operations.

The Forescout Research dubbed Sierra:21 discovered 71% percent of vulnerabilities (15 of 21) were open-source software components – and most went unreported. Given that discovery, we wanted to understand what that meant for a much broader range of devices across many vendors.

We began by identifying 290 firmware images from 39 manufacturers of OT/IoT network equipment available on the internet. We narrowed this list to include only devices that function as cellular routers, have firmware images that can be automatically analyzed, and are relatively popular (based on results from the Shodan search engine).

From this refined list, we selected five firmware images for analysis, as shown in Table 1. The table details the specific models and firmware versions analyzed, alongside the number of exposed devices from each vendor found via Shodan. All the firmware versions analyzed were the latest we could find online at the time of the research (January 17, 2024).

Table 1 – Selected firmware images and hits on Shodan

Vendor	Model(s)	Version (Release Date)	Operating System (OS)	#Hits on Shodan (query)
Acksys	AirBox LTE, AirWan-M12, AirLink, WaveNet-Ex	4.22.3.1 (Jan/2024)	WaveOS	26 (support@acksys.fr OR http.html:Acksys)
Digi	IX10	23.9.20.67 (Dec/2023)	DAL OS	3,685 (http.favicon.hash:931954617)
MDEX	MX880	02.517 (Oct/2022)	RutOS	316 (http.favicon.hash:-1964821744)
Teltonika	RUT950	00.07.06.1 (May/2023)	RutOS	70,959 (http.title:"teltonika")
Unitronics	UCR	51.06.06.185 (Sept/2021)	RutOS	206 (http.favicon.hash:1683669412)

Source: Forescout Vedere Labs

Figure 1 illustrates Vedere Labs analysis methodology. Each firmware image was uploaded into the Finite State platform to gather data on software composition, historical vulnerabilities and public exploits that might affect the identified software components. Additionally, we collected data on binary hardening practices within the selected firmware images using a separate custom script based on checksec.sh. Finally, we manually validated the reported results to distinguish accurate components and vulnerabilities from false positives, which are inevitable in automated analysis due to the context-dependent nature of vulnerabilities For example, CVE-2018-1000517 affects *busybox*, but not the way that component is used in Digi's firmware, as the binary in that firmware does not include the vulnerable *busybox* applet *wget*.

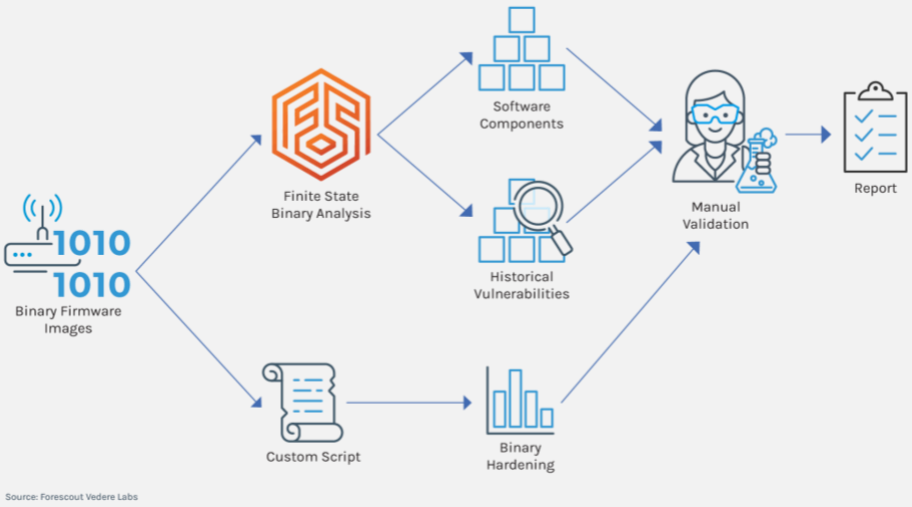


Figure 1 – Analysis methodology

3. Main Findings

3.1. OpenWrt Is Everywhere

After an initial analysis of the five selected firmware image filesystems, we found that all except Digi run operating systems derived from OpenWrt, an open-source Linux-based OS for embedded devices. The OpenWrt project started in 2004, was originally based on open-source components of the Linksys WRT54G router. The main components of the project include the Linux kernel, util-linux, musl and buysbox, with over 8,000 additional software packages available through the opkg package management system.

OpenWrt is well-known among hobbyists who customize their routers, supporting more than 2,200 different router models. However, it also serves as the foundation for several commercial firmware images, including those from Table 1, as well as Ubiquiti, TP-Link, Xiaomi, ZyXEL, and D-Link. Some vendors use other OSes based on OpenWrt. For instance, Teltonika and Acksys have developed in-house versions of OpenWrt called RutOS and WaveOS, while MDEX and Unitronics run their own modifications of RutOS. The only OS in Table 1 not related to OpenWrt is DAL OS (Digi Accelerated Linux), used by Digi.

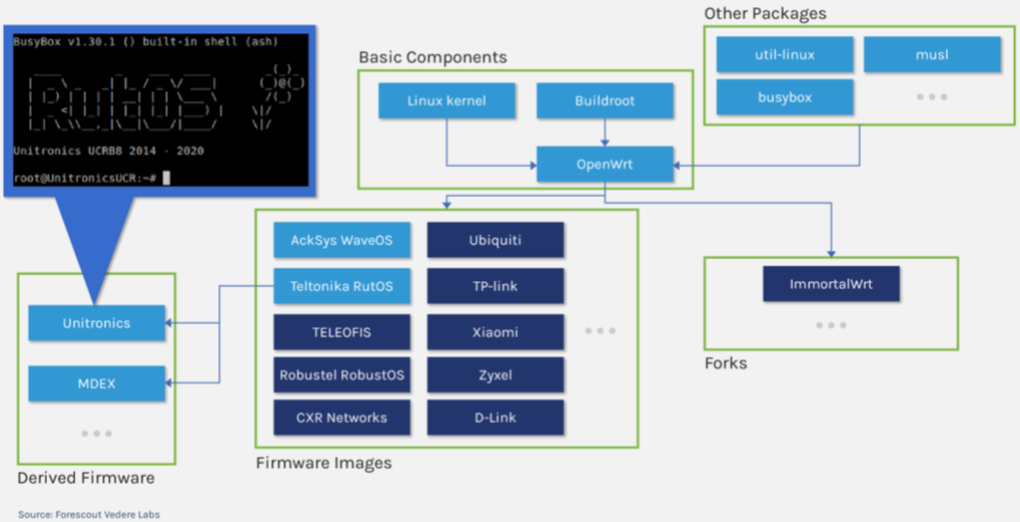


Figure 2 – OpenWrt supply chain and Unitronics router showing the RutOS banner originally from Teltonika

Figure 2 summarizes this supply chain. OpenWrt combines basic components, such as the Linux kernel with various *other packages*, which are then used in the *firmware images* of several vendors. Some vendors create *derived firmware* based on the original OpenWrt, while other projects, such as ImmortalWrt, fork the original OpenWrt for their own purposes. Components highlighted in blue in the figure are part of this research, whereas others are out of scope, but illustrate the widespread use of OpenWrt. The figure also shows the login banner of the Unitronics router displaying the "RutOS" message originally from Teltonika routers.

As is common with long software supply chains, there are risks of vulnerabilities not being patched downstream or emerging due to how components are integrated. For example, TP-Link SOHO routers have known vulnerabilities exploited in the wild, originating from the LuCI web application component of OpenWrt: CVE-2017-16959 and CVE-2023-1389. The latter is frequently observed on Forescout's AEE being used to deploy Mirai botnet variants.

To understand *how* the four selected firmware images utilize the base OpenWrt OS, we compared the file hashes of binaries in each firmware, as identified by Finite State, with the hashes of binary files from five major OpenWrt releases (17.*, 18.*, 19.*, 21.*, 22.*, 23.*). Figure 3 shows that none of the firmware, had more than 15% of their components matching known hashes from any given OpenWrt version. This indicates that these firmware images use heavily modified versions of the base OpenWrt OS, either by mixing and matching individual component versions with a base version or by developing their own in-house components.

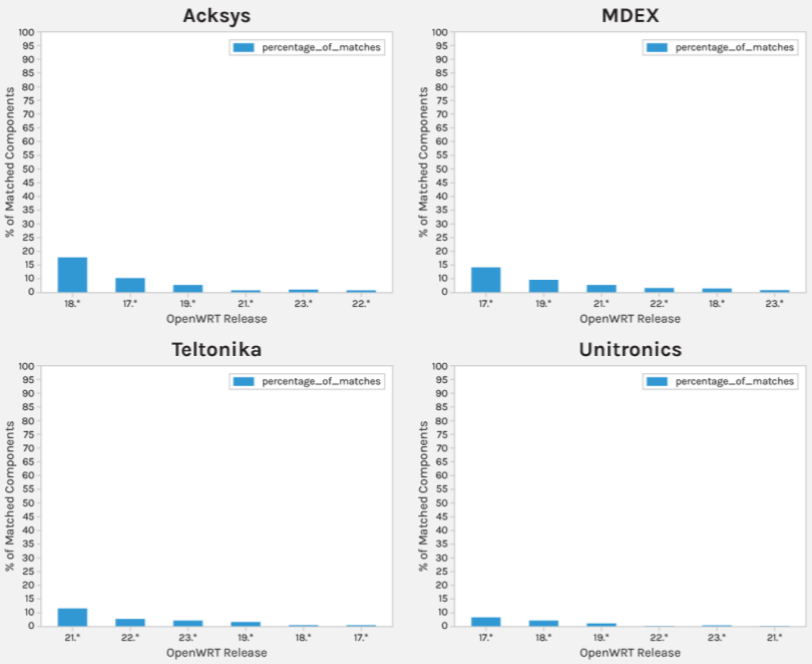


Figure 3 – Matches between binaries in each firmware image and OpenWrt base versions

3.2. Software Components Are Often Outdated

When we uploaded the five selected firmware images to the Finite State platform, the tool returned the results summarized in Table 2. The analysis time ranged from 35 minutes to 1 hour 45 minutes per firmware (averaging 1 hour 19 minutes). It identified between 500 and nearly 900 components in each firmware (an average of 662 components), and between 1,200 and 2,500 "findings" (an average of 2,154). Findings included known vulnerabilities, weak security posture (such as default credentials or hardcoded cryptographic material) and potential new vulnerabilities identified through binary static analysis. On average, the automatic analysis identified more than three findings per component across all firmware images.

Table 2 – Summary of components and findings from the Finite State platform

Firmware	Components	Findings	Average Findings per Component	Processing Time
Acksys	509	2500	4.9	1h 42 min
Digi	879	2059	2.3	1h 31 min
MDEX	449	2509	5.6	0h 35 min
Teltonika	841	1283	1.5	1h 27 min
Unitronics	636	2422	3.8	1h 20 min
Average	662.8	2154.6	3.6	1h 19 min

Source: Forescout Vedere Labs

The Finite State platform also categorized the findings by severity (low, medium, high and critical) and assigned a risk score to each, as well as a summarized risk score for each firmware image, as shown in Figure 4 (captured from the tool's user interface). Most findings were categorized as low severity, but every firmware had at least one critical finding. On average, a firmware image had a risk score of 84.4, with 7 critical findings, 33.4 high-severity findings, 212.6 medium-severity findings and 1901.6 low-severity findings.

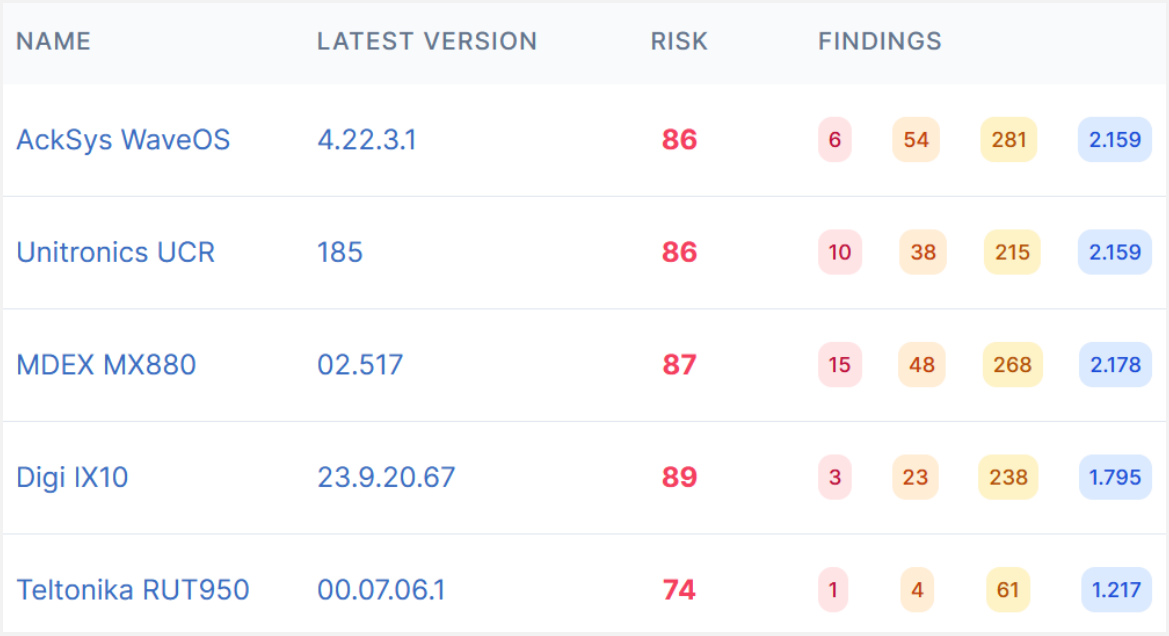


Figure 4 – Categorized findings and risk score provided by the Finite State platform

Since manually verifying and enriching each component or finding is beyond the scope of this research, we focused on the components that are most common across firmware images (used in at least three of the five images in question) and have known vulnerabilities.

Table 4 (in the appendix) lists the 25 selected open-source software components. Publicly available exploits exist for at least one version of 12 out of these 25 components. Keep reading for more insight on n-day vulnerabilities. The table provides insights into the open-source components used by different firmware images and its state of software component decay:

- Out of 25 components, only one is used in its latest version by every firmware: shellinabox (2016)
- lilblzo is used in its latest version by every image except MDEX
- None of the five firmware images has the latest versions of all components

- Digi has the latest version of three components:
  - busybox, dnsmasq and libpcre
  - Teltonika has the latest versions of two (gpsd and hostapd)

- The other three firmware images have all outdated components.
  - The average open-source component on an OT/IoT router is:
    - 5 years and 6 months old
    - 4 years and 4 months behind the latest release

- The oldest component was readline 6.2 (13 years old)
- The furthest behind the latest release was ncurses 5.9 (more than 11 years behind ncurses 6.4)

- No firmware used the latest Linux kernel version:
  - The oldest kernel was Linux 3.10.36 on MDEX
    - It was released in April 2014
    - making it almost 10 years old at the time of investigation
  - The newest kernel was Linux 6.3 on Digi
    - Released in April 2023
    - More than 6 months before the firmware release

Overall, even the most recent firmware images often do not use the latest release of open-source components, including some of the most critical ones. Notably, only Teltonika uses a Linux kernel version that is still maintained. The kernels used by Digi and Acksys have been unsupported since 2023, while those used by MDEX and Unitronics reached end of life in 2017. Additionally, all firmware images, except for Digi's, rely on an unsupported version of OpenSSL.

Another important observation is that although the Digi device performs a similar function to the others and has no relation to OpenWrt, Table 4 shows that most embedded Linux systems are quite similar. If two devices from two different vendors are designed to perform similar functions, they are likely built using the same open-source components.

3.3. Known Vulnerabilities Abound

Returning to the discussion on component decay from the previous section, Figure 5 shows the average and maximum "age" of components for each firmware in months. Here, "age" refers to the time elapsed between the release date of a component and the date of the research (January 17, 2024). Digi has the newest components on average, followed by Teltonika, Acksys, Unitronics and MDEX.

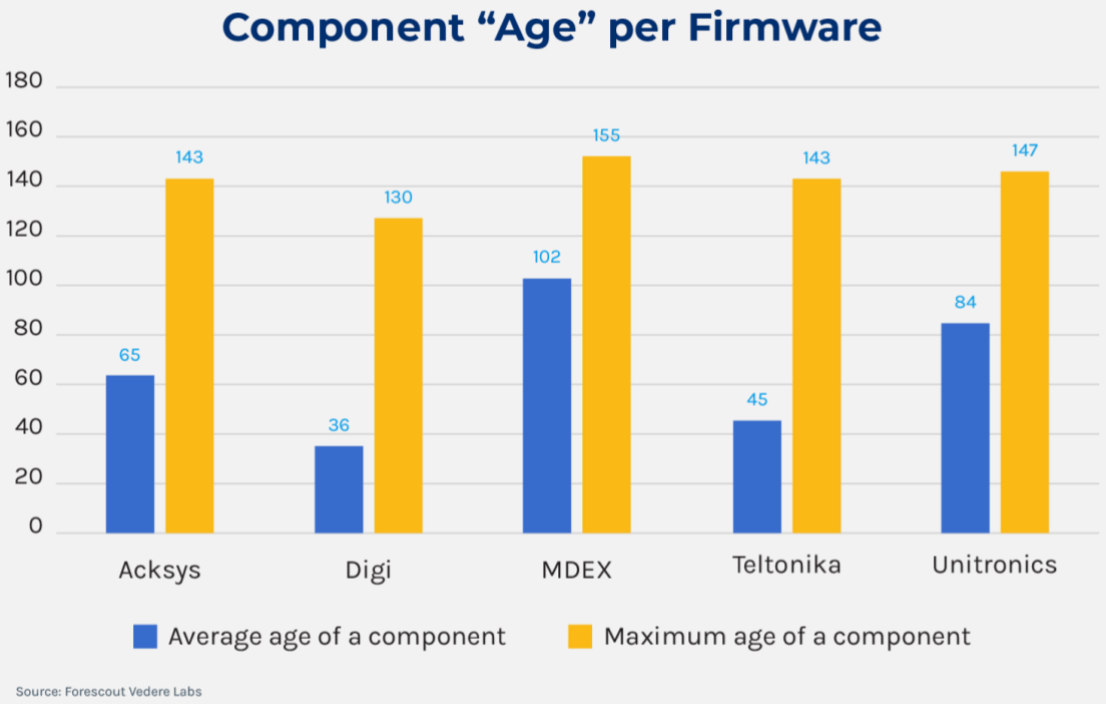


Figure 5 - Component age per firmware

To determine whether the age of components directly affects the number of historical (or "n-day") vulnerabilities present, we examined the total number of vulnerabilities per component in each firmware.

We separated the Linux kernel from other third-party components (mostly Linux packages) when analyzing historical vulnerabilities due to the high volume of issues in the kernel. We will first discuss the third-party components and then address the kernel issues.

Third-Party Components

Figure 6 shows the number of historical vulnerabilities associated with the common open-source components used in the firmware images we analyzed. On average, these firmware images had 161 known vulnerabilities in their most common components. It is evident that Teltonika and Digi have accumulated the fewest vulnerabilities, followed by Acksys, Unitronics and MDEX. This is unsurprising, as their open-source package versions are more up-to-date compared to the other firmware images on our list. The distribution of vulnerabilities is almost perfectly correlated with the age of software components, with the exception that Teltonika has fewer vulnerabilities than Digi.

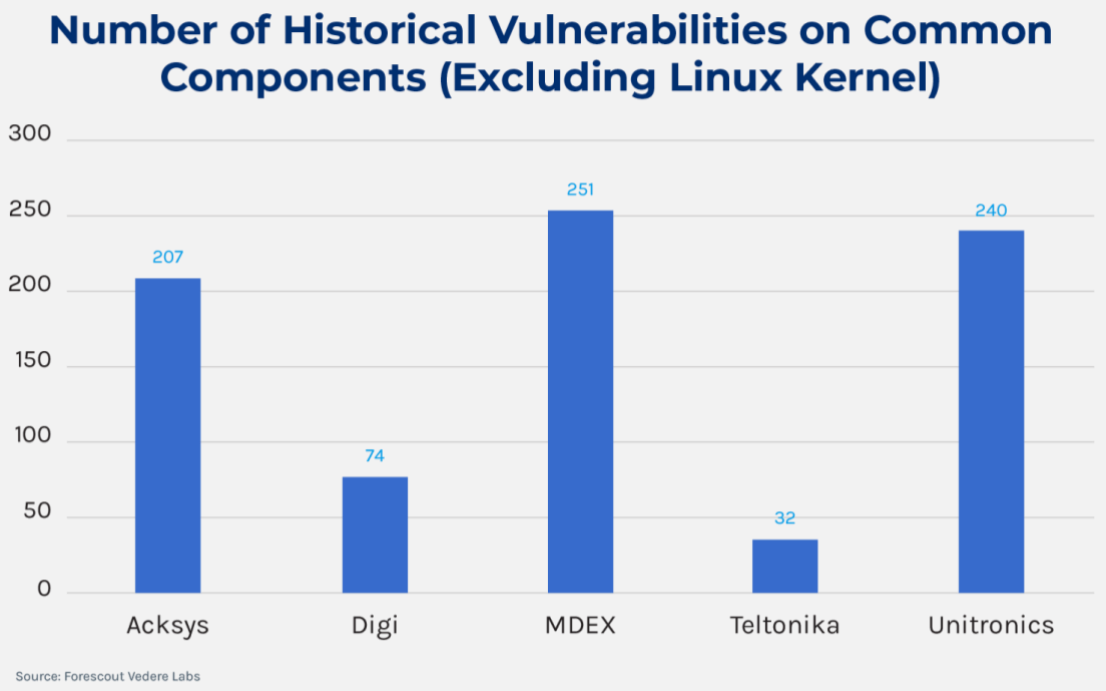


Figure 6 – Historical vulnerabilities in the analyzed firmware images

Figure 7 breaks down the historical vulnerabilities by CVSSv3 score. On average, firmware images had 68 vulnerabilities with a low or medium score, 69 with a high score and 24 with a critical score. Digi and Teltonika have relatively few vulnerabilities in their open-source components with the highest score being 9.8 (7 and 6 vulnerabilities respectively). In contrast, Unitronics, MDEX, and Acksys have 21, 24, and 27 with high scores, respectively.

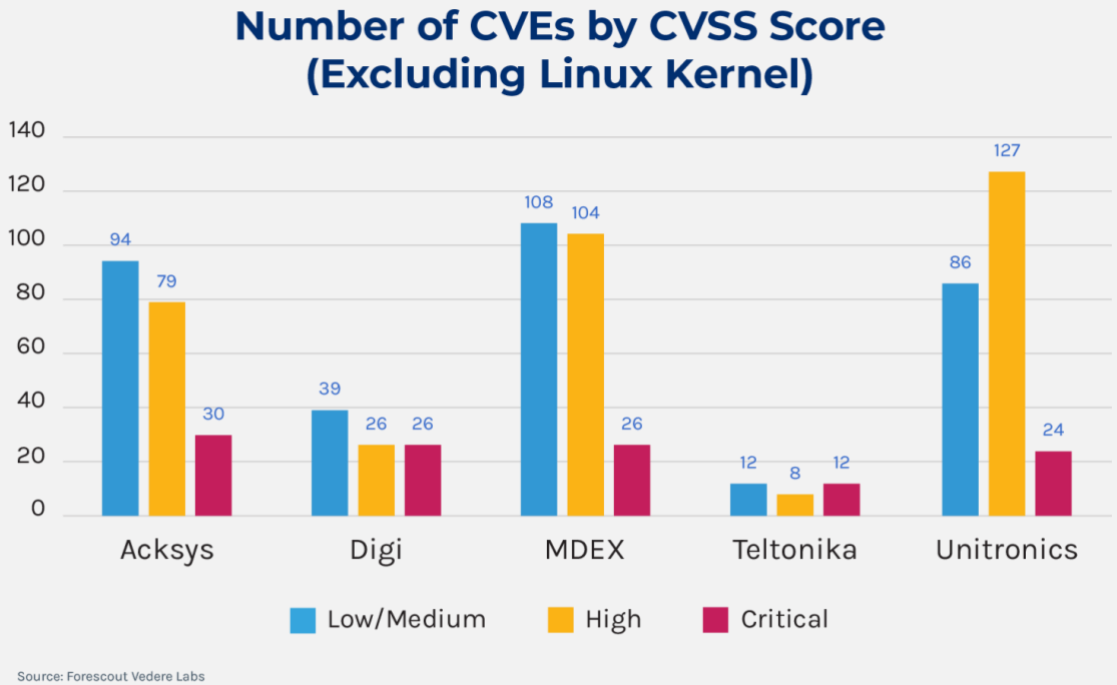


Figure 7 – Number of historical vulnerabilities by CVSS score

Linux Kernel

Table 5 (in the appendix) lists the n-day vulnerabilities affecting the Linux kernel used in the firmware images we analyzed. Given the thousands of vulnerabilities associated with these kernel versions, we narrowed our analysis to those vulnerabilities that allow for privilege escalation or remote code execution and have a public exploit or detailed write-up, as these can more easily be used to take over a vulnerable device,

To estimate whether a kernel version is affected by a vulnerability, we consulted the Common Platform Enumeration (CPE) data on the National Vulnerability Database (NVD) and cross-checked the release dates of the known affected versions. We also excluded vulnerabilities listed as fixed in security advisories from the vendors (if available). We found firmware changelogs from Acksys, Digi, MDEX, and Teltonika, as well as dedicated security advisories from Acksys (currently offline), Digi, Teltonika, and Unitronics. However, these advisories contained very few mentions of patches applied to the Linux kernel. This lack of detail is understandable for current LTS kernels that should incorporate all patches (as is the case with Teltonika) but it raises concerns for older, unsupported kernel versions.

On average, the firmware images had 20 exploitable n-day vulnerabilities affecting the kernel. Some notable observations for specific firmware include:

- Teltonika runs the latest version of an LTS kernel (5.4.259) so it was not affected by any exploits.
- Digi uses a relatively fresh kernel (6.3.0), but its support ended in July 2023 and several exploits have emerged since then.
- Although the kernel releases of Acksys, MDEX, and Unitronics, are several years apart, they are affected by roughly the same number of exploits (between 29 and 33). This similarity may be explained by the ever-changing nature of the Linux kernel project: with newer releases, old vulnerabilities are eventually patched, but new functionality often introduces new issues.

3.4. Security Features Are Lacking

We evaluated the security practices of the firmware images in **two ways**: the use of binary hardening techniques, which make exploitation of vulnerabilities more difficult, and the use of default credentials, which can allow attackers to take over devices if the credentials are not changed or are easy to guess.

Binary Hardening

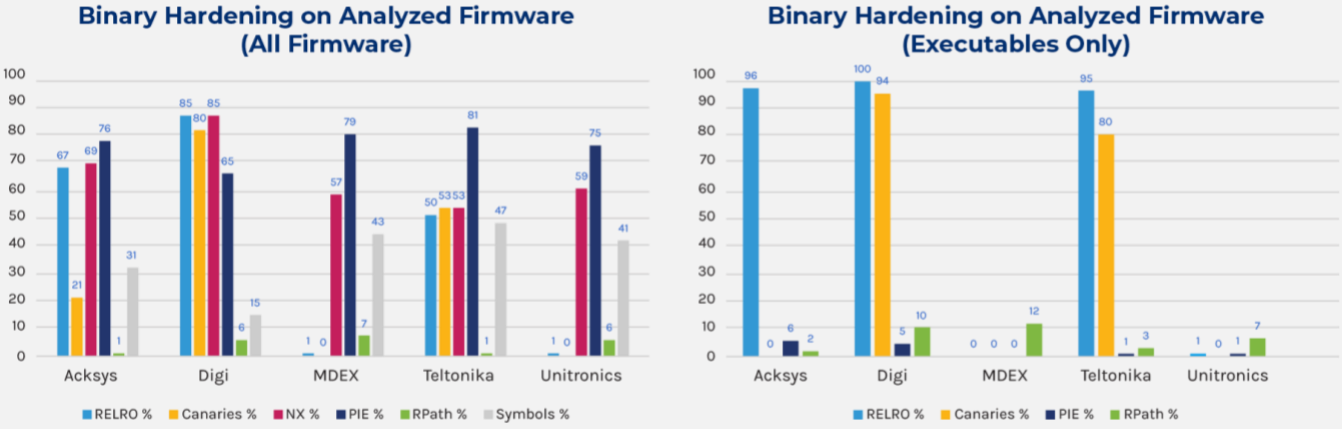
To assess binary hardening practices, we considered the following features:

- **RELRO** (Relocation Read-Only): This feature ensures that the linker resolves all dynamically-linked functions at the beginning of the program execution and sets the Global Offset Table (GOT), used for resolving function addresses in shared libraries, to read-only. This prevents attackers from arbitrarily redirecting execution by overwriting the GOT.
- **PIE** (Position Independent Executable): This mechanism loads all functions of the target binary and its dependencies into arbitrary locations in virtual memory, making it much more difficult to launch Return-Oriented Programming (ROP) attacks.
- **Stack canaries**: These are secret, randomly generated, values placed onto the stack of a function. Before a function returns, the canary value is checked and, if it has been altered, the program exits. This prevents straightforward stack smashing exploits, although it does not prevent scenarios when stack canaries may be “leaked” to the attacker.
- **NX** (No eXecute bit): This hardware feature allows an operating system to mark certain memory pages as non-executable, preventing code from running in these areas. When used, the stack and heap areas are marked as non-executable, preventing straightforward buffer overflow exploits.
- **RPath** (Runtime library search path): This feature in Unix-like systems informs the dynamic linker of the location of the shared libraries. While it can improve portability and compatibility, it must be used carefully, as including relative or untrusted paths that point to shared libraries introduces security risks.

- **Symbols in binaries**: Symbols are used for debugging purposes but are also aid attackers, by simplifying reverse engineering and exploitation. Since most binaries in question are open source, the presence of debugging symbols does not give a significant advantage.

RELRO has a minor performance impact when program execution starts, while PIE affects both startup times and runtime performance. When these mechanisms are combined, the performance overhead may become more noticeable in certain cases. There are different variants of stack canaries, some of which are relatively expensive in terms of performance overhead. However, any overheads introduced by binary hardening are generally considered acceptable due to their added security benefits, especially for network devices. We demonstrated the effect of missing hardening features in Vedere Labs previous research, Sierra:21.

The left-hand side of Figure 8 displays the percentage of all binaries, including executables, shared libraries, and kernel modules, within each analyzed firmware that have binary hardening features present. On average, 41% of binaries across firmware images use RELRO, 31% use stack canaries, 65% use NX, 75% use PIE, 4% use RPath and 35% have debugging symbols. MDEX and Unitronics appear extremely similar, likely because both derive from Teltonika's RutOS, and their release dates are relatively close. Teltonika's firmware looks notably different, likely because it was released much later than the former two with changes that may have not yet been incorporated into the customized variants of RutOS used by MDEX and Unitronics.



Source: Forescout Vedere Labs

Figure 8 – Binary hardening in the analyzed firmware images (all binary files)

The right-hand side of Figure 8 shows the same information, but focuses only on executables:

- The "NX" and "symbols" bars are missing because the flag is enabled in all executables and they have all been stripped, preventing a whole class of trivial buffer overflow exploits.
- On average, 58% of executables use RELRO, 35% use stack canaries, only 2.5% use PIE and only 7% use RPath. However, these averages can be misleading due to significant differences between firmware images, which we explore below.
- Most executables in Teltonika and Digi use stack canaries, while most executables in Acksys, Teltonika and Digi have RELRO enabled, ruling out attacks involving GOT overwrites.
- Most executables across all firmware images do not use PIE, introducing significant security risks, especially for executables that do not use stack canaries (MDEX, Unitronics, and Acksys). The effectiveness of Address-space Layout Randomization (ASLR) is significantly limited when PIE is not used, and the absence of stack canaries makes it even easier to implement buffer overflow exploits.
- In summary, if a buffer overflow is found in almost any of the executables within Acksys, MDEX and Unitronics, it is highly likely that it can be exploited to achieve code execution. Not surprisingly, Teltonika and Digi, the vendors with the most recent components and fewer n-days are also the ones best employing binary hardening techniques.

Default Credentials

Table 3 shows how the analyzed firmware images use default credentials. All of them come with pre-configured default credentials, derived in different ways. This would be a problem if they did not force password changes on the first use. We confirmed that four out of five firmware images do require this change. For the fifth image, Acksys, we could not confirm whether the change is forced or not, so it is marked as "No" with an asterisk in the table. Overall, these results indicate that modern firmware for OT/IoT routers generally do not suffer from the issue of easily exploitable default credentials.

Table 3 – Use of default credentials on the analyzed firmware

Firmware	Default Credentials?	Is Change Forced?
Acksys	Yes (empty)	No*
Digi	Yes (unique, factory-assigned)	Yes
MDEX	Yes (based on serial number)	Yes
Teltonika	Yes	Yes
Unitronics	Yes	Yes

Source: Forescout Vedere Labs

4. Technical Deep Dive: Custom Security Patches

Sometimes downstream vendors choose to provide *their own support* for some third-party components, applying upstream security patches or fixing vulnerabilities independently of the original maintainers. Various reasons may drive this choice, such as a component no longer being maintained upstream or a component being difficult to upgrade without breaking important functionality. We often encountered this during Project Memoria, where embedded device vendors used end-of-life TCP/IP stacks like uIP or operating systems like Contiki in still supported products

This approach can lead to confusion, especially when there are no public records or security advisories mentioning these custom patches. Furthermore, some custom patches may be incorrect, not only failing to remediate a vulnerability, but potentially introducing new ones due to incomplete patching.

To illustrate this situation, consider the example of CVE-2020-8597, a stack buffer overflow in the pppd daemon used by four of the five firmware analyzed in this research. The vulnerability can be triggered by sending an unsolicited Extensible Authentication Protocol (EAP) MD5-Challenge message to a vulnerable PPP server or client, resulting in a denial of service or remote code execution.

Figure 9 shows a fragment of the official patch for CVE-2020-8597. Two functions that parse EAP requests and responses – *eap_request()* and *eap_response()* – contained the same vulnerable code. Therefore, we only show the fragment for the eap_request() function. One way to trigger the vulnerability is by sending a long peer name as part of an EAP request. Here, *rhostname* is a buffer that is supposed to hold a peer name (an array of 256 bytes), *vallen* (unsigned char) is the length of the corresponding tag (a specific field in a packet), and *len*

(integer) is the actual length of the peer name. It is possible to craft an EAP request, so that the value of *vallen* becomes smaller than *len+256* (i.e., *sizeof(rhostname)*); at the same time, *vallen* should be larger than *len*. This will ensure that the "else" branch of the condition shown on line 1432 will be executed, overflowing the *rhostname* buffer.

```
1414                if (vallen < 8 || vallen > len) {
1415                        error("EAP: MD5-Challenge with bad length %d (8..%d)",
1416                            vallen, len);
1417                        /* Try something better. */
1418                        eap_send_nak(esp, id, EAPT_SRP);
1419                        break;
1420                }
1421
1422                /* Not so likely to happen. */
1423        -       if (vallen >= len + sizeof (rhostname)) {
1424                        dbglog("EAP: trimming really long peer name down");
1425                        BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
1426                        rhostname[sizeof (rhostname) - 1] = '\0';
1427                } else {
1428                        BCOPY(inp + vallen, rhostname, len - vallen);
1429                        rhostname[len - vallen] = '\0';
1430                }
```

```
1414                if (vallen < 8 || vallen > len) {
1415                        error("EAP: MD5-Challenge with bad length %d (8..%d)",
1416                            vallen, len);
1417                        /* Try something better. */
1418                        eap_send_nak(esp, id, EAPT_SRP);
1419                        break;
1420                }
1421
1422                /* Not so likely to happen. */
1423        +       if (len - vallen >= sizeof (rhostname)) {
1424                        dbglog("EAP: trimming really long peer name down");
1425                        BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
1426                        rhostname[sizeof (rhostname) - 1] = '\0';
1427                } else {
1428                        BCOPY(inp + vallen, rhostname, len - vallen);
1429                        rhostname[len - vallen] = '\0';
1430                }
```

Figure 9 – The patch for CVE-2020-8597

According to the original vulnerability description, CVE-2020-8597 affects pppd versions from 2.4.2 through 2.4.8. Based on this information, three of the firmware images that we analyzed contain a vulnerable binary (see Table 4): Teltonika, MDEX and Unitronics. To confirm this, we analyzed the **pppd** binary shipped with these firmware images and found two distinct situations:

1. **MDEX and Teltonika**: Both contained the official patch. However, their version numbers remained unchanged: 2.4.7 and 2.4.8, respectively. Additionally, the corresponding changelog advisories from both vendors clearly stated that CVE-2020-8597 was patched.

2. **Unitronics**: This firmware contained a *custom and incorrect* patch for CVE-2020-8597. Figure 10 shows a pseudocode fragment generated from the corresponding disassembly. The entire condition that checks *len* and *vallen* has been removed (line 1423 on Figure 9), along with the contents of the "if" branch, leaving the unguarded vulnerable "else" branch in place. We checked the official pppd source code repository and concluded that these changes are a custom patch rather than an older unpatched version. We also emulated the pppd binary on the firmware image and confirmed that the vulnerability could be triggered, indicating that the patch was incorrect.

```
 95      len_2 = len;
 96      if ( vallen < 8 || len < (int)vallen )
 97      {
 98        error("EAP: MD5-Challenge with bad length %d (8..%d)", vallen, len_2);
 99LABEL_34:
100        result = sub_418498(a1, a3);
101        goto LABEL_35;
102      }
103      v31 = (unsigned __int8 *)(a2 + 2);
104      v30 = len_2 - vallen;
105      memcpy(rhostname, &a2[vallen + 2], len_2 - vallen);// BCOPY
```

Figure 10 – A vulnerable pseudocode fragment in the pppd binary from Unitronics

We could not find any changelog or security advisory mentioning this custom patch from Unitronics. This example illustrates a likely case of silent patching that may leave a product exposed for longer, as the vendor has a false impression that the vulnerability has been patched and no further action is necessary.

Note: after concluding our analysis of the latest Unitronics firmware image available at the time of this research, we discovered that on February 12th 2024, two newer versions of Unitronics firmware images were uploaded (versions 17.01.12.25 and 51.06.06.252). We checked the pppd binary in these images and they now appear to contain the same (correct) patch seen in Teltonika and MDEX.

5. Conclusion

This research reveals that when analyzing firmware images from different vendors for a similar category of devices they exhibit both:

- **Similarity in component usage**: Many devices rely on the same operating system distributions and optional packages.
- **Differences in component versions**: The specific versions of these components vary significantly. Notably, a more recent firmware from vendor A does not necessarily have more up-to-date components than an older firmware from vendor B. Importantly, there is a correlation between using newer versions of software components and having fewer vulnerabilities, as well as better security practices.

Even minor differences in software versions can significantly impact vulnerabilities and, consequently, the risk that a device poses to an asset owner. Because the current system of changelogs and security advisories fails to provide adequate information about which vulnerabilities affect a device, **this research provides further empirical evidence for the need for precise SBOMs**.

SBOMs can be generated and provided by the device manufacturers or reconstructed from a firmware image, as we did throughout this research using Finite State. However, most manufacturers still decline to provide SBOMs to their customers and those that do often take months to deliver outdated or incomplete information. Therefore, automated SBOM generation from firmware is necessary, but it is inherently limited due to factors like encrypted firmware, proprietary formats, and exotic CPU architectures.

SBOM solutions are more effective when used to enrich information about vulnerabilities, exploits, threats and risks on an initial SBOM provided by the manufacturer. They are also used to identify inconsistencies in that provided SBOM rather than build an SBOM from scratch.

Device manufacturers need to improve the information provided to customers. They should be more transparent about the components they use, the versions of these components and the patches that have been applied. And they should use standardized formats whenever possible. This transparency helps customers understand their risks, mitigate them quickly and use the information in an interoperable way with other solutions. It also demonstrates a manufacturer's maturity, as security through obscurity never works.

The number of vulnerabilities is expected to grow. **On top of increasing lines of code leading to more vulnerabilities, the very understanding of what constitutes a vulnerability is expanding**. For example, in February 2024, the Linux kernel project became a CVE numbering authority (CNA) and will assign a CVE to every bug in the kernel. As more attackers target embedded devices, the risk to these devices is also increasing.

SBOMs and precise component information are crucial not only when procuring new equipment or planning patching, but also for understanding device risk and detecting and responding to threats on the network. Effective OT/IoT risk mitigation requires a network-based, granular and dynamic asset inventory. This inventory should be capable of extracting information such as vendor, OS, firmware version, and more, using passive network fingerprints or active capabilities. It should also correlate that information with SBOMs to provide contextualized risk information and support the enforcement of mitigating controls.

Appendix – Data Tables

Table 4 – The most common open-source components (cells in red represent available exploits)

Component latest release	Acksys (04.01.2024)	Digi (01.12.2023)	MDEX (01.10.2022)	Teltonika (19.05.2023)	Unitronics (14.09.2021)
Linux 6.6.9 (1-1-2024)	Linux 4.9.111 (3-Jul-2018)	Linux 6.3.0 (23-Apr-2023)	Linux 3.10.36 (3-Apr-2014)	Linux 5.4.259 (25-Oct-2023)	Linux 3.18.44 (24-Oct-2016)
OpenSSL 3.2.0 / 1.1.1w (23-Nov-2023) / (12-Sep-2023)	OpenSSL 1.1.1n (15-Mar-2022)	OpenSSL 3.1.1 (30-May-2023)	OpenSSL 1.1.1k (25-Mar-2021)	OpenSSL 1.1.1t (07-Feb-2023)	OpenSSL 1.1.1c (28-May-2019)
busybox 1.36.1 (19-May-2023)	busybox 1.28.3 (3-Apr-2018)	busybox 1.36.1 (19-May-2023)	busybox 1.30.1 (10-June-2019)	busybox 1.34.1 (30-Sep-2021)	busybox 1.30.1 (10-June-2019)
curl 8.5.0 (6-Dec-2023)	curl 7.60.0 (16-May-2018)	curl 8.1.2 (30-May-2023)	curl 7.66.0 (11-Sep-2019)	curl 8.4.0 (11-Oct-2023)	curl 7.64.0 (6-Feb-2019)
dnsmasq 2.89 (4-Feb-2023)	dnsmasq 2.80 (18-10-2018)	dnsmasq 2.89 (4-Apr-2023)	dnsmasq 2.82 (19-Jul-2020)	dnsmasq 2.85 (7-Apr-2021)	dnsmasq 2.79 (18-Mar-2018)
libgps / gpsd 3.25 (10-Jan-2023)	gpsd 3.17 (7-Sep-2017)	gpsd 3.21 (4-Aug-2020)	gpsd 3.0 (3-Oct-2011)	gpsd 3.25 (10-Jan-2023)	gpsd 3.0 (3-Oct-2011)
hostapd 2.10 / wpa_supplicant 2.10 (16-Jan-2021)	hostapd 2.10 / wpa_supplicant 2.10 (16-Jan-2021)	hostapd 2.10 / wpa_supplicant 2.10 (16-Jan-2021)	hostapd 2.7 / wpa_supplicant 2.7 (2-Dec-2018)	hostapd 2.10 / wpa_supplicant 2.10 (16-Jan-2021)	hostapd 2.7 / wpa_supplicant 2.7 (2-Dec-2018)
iptables 1.8.10 (10-Oct-2023)	iptables 1.6.2 (2-Feb-2018)	iptables 1.8.7 (15-Jan-2021)	iptables 1.4.21 (22-Nov-2013)	iptables 1.8.7 (15-Jan-2021)	iptables 1.4.21 (22-Nov-2013)
libpcap 1.10.4 (7-Apr-2023)	libpcap 1.9.0 (24-Jun-2018)	libpcap 1.10.0 (29-Dec-2020)	libpcap 1.5.3 (18-Dec-2013)	libpcap 1.9.1 (22-Jul-2018)	libpcap 1.8.1 (26-Oct-2016)
zlib 1.3.0 (18-Aug-2023)	zlib 1.2.11 (15-Jan-2017)	zlib 1.2.11 (15-Jan-2017)	zlib 1.2.8 (29-Apr-2013)	zlib 1.2.11 (15-Jan-2017)	zlib 1.2.11 (15-Jan-2017)
strongswan 5.9.13 (1-Dec-2023)	strongswan 5.9.5 (24-Jan-2022)	strongswan 5.9.10 (2-Mar-2023)	strongswan 5.3.3 (7-Sep-2015)	strongswan 5.9.2 (26-Feb-2021)	strongswan 5.6.2 (19-Feb-2018)
openvpn 2.6.8 (17-Nov-2023)	openvpn 2.4.9 (17-Apr-2020)	openvpn 2.6.4 (11-May-2023)	openvpn 2.4.7 (20-Feb-2019)	openvpn 2.5.3 (17-Jun-2021)	openvpn 2.4.5 (6-Apr-2018)
pppd 2.5.0 (4-Apr-2023)	N/A	pppd 2.4.9 (4-Jan-2021)	pppd 2.4.7 (9-Aug-2014)	pppd 2.4.8 (21-Mar-2020)	pppd 2.4.7 (9-Aug-2014)
libpcre 8.45 / libpcre2 10.39 (22-Jun-2021) EoL / (29-Oct-2021)	N/A	libpcre 8.45 (22-Jun-2021)	libpcre 8.35 (8-Apr-2014)	libpcre2 10.37 (26-May-2021)	libpcre 8.42 (2-Apr-2018)
util-linux 2.39.3 (4-Dec-2023)	N/A	util-linux 2.37.2 (16-Aug-2021)	util-linux 2.24.1 (20-Jan-2014)	util-linux 2.36.1 (16-Nov-2020)	util-linux 2.24.1 (20-Jan-2014)
lua 5.4.6 (2-May-2023)	lua 5.1.5 (13-Feb-2012)	N/A	lua 5.1.5 (13-Feb-2012)	lua 5.1.5 (13-Feb-2012)	lua 5.1.5 (13-Feb-2012)

liblzo 2.10 (1-Mar-2017)	liblzo 2.10 (1-Mar-2017)	N/A	liblzo 2.06 (12-Aug-2011)	liblzo 2.10 (1-Mar-2017)	liblzo 2.10 (1-Mar-2017)
dropbear 2022.83 (14-Nov-2022)	dropbear 2017.75 (18-May-2017)	N/A	dropbear 2018.76 (27-Feb-2018)	dropbear 2020.81 (29-Oct-2020)	dropbear 2019.78 (27-Mar-2019)
ncurses 6.4 (31-Dec-2022)	ncurses 6.1 (27-Jan-2018)	ncurses 6.3 (8-Nov-2021)	ncurses 5.9 (4-Apr-2011)	N/A	ncurses 6.1 (27-Jan-2018)
readline 8.2 (26-Sep-2022)	readline 7.0 (15-Sep-2016)	readline 7.0 (15-Sep-2016)	readline 6.2 (13-Feb-2011)	N/A	readline 7.0 (15-Sep-2016)
tcpdump 4.99.4 (7-Apr-2023)	tcpdump 4.9.2 (3-Sep-2017)	tcpdump 4.99.1 (9-Jun-2021)	tcpdump 4.9.3 (30-Sep-2019)	N/A	tcpdump 4.9.2 (3-Sep-2017)
musl libc 1.2.4 (1-5-2023)	musl libc 1.2.3 (7-Apr-2022)	musl libc 1.2.3 (7-Apr-2022)	N/A	musl libc 1.1.24 (13-Oct-2019)	N/A
shellinabox 2.20 (9-11-2016)	N/A	shellinabox 2.20 (9-11-2016)	N/A	shellinabox 2.20 (9-11-2016)	shellinabox 2.20 (9-11-2016)
stunnel 5.71 (19-Sep-2023)	N/A	stunnel 5.66 (11-Sep-2022)	N/A	stunnel 5.50 (2-Dec-2018)	stunnel 5.50 (2-Dec-2018)
libffi 3.4.4 (24-Oct-2022)	libffi 3.3-2 (23-Nov-2019)	libffi 3.0.13 (17-Mar-2013)	libffi 3.0.13 (17-Mar-2013)	N/A	N/A

Table 5 – Linux kernel exploits (cells in red represent vulnerabilities with available exploits affecting a specific firmware)

CVE	Exploit / writeup link	Acksys (29)	Digi (6)	MDEX (33)	Teltonika (0)	Unitronics (31)
CVE-2023-2598	io_uring out-of-bounds access to physical memory					
CVE-2023-3269	Use-after-free in Linux memory management subsystem (StackRot)					
CVE-2023-3389	Use-after-free in io_uring (LinkedPoll)					
CVE-2022-0435	A stack overflow flaw was found in the Linux kernel's TIPC protocol					
CVE-2022-34918	A heap-based buffer overflow in Netfilter					
CVE-2022-32250	Use-after-free in Netfilter					
CVE-2023-32233	Netfilter nf_tables use-after-free					
CVE-2023-3390	Use-after-free in Netfilter					
CVE-2023-35001	Out-of-bounds read/write in nftables					
CVE-2022-27666	A heap-based buffer overflow in IPSec ESP					
CVE-2022-1015	Out-of-bounds read/write in netfilter subsystem					
CVE-2022-29582	Use-after-free in io_uring					
CVE-2022-2586	Linux kernel nft_object use-after-free					
CVE-2022-42703	Use-after-free in the Linux memory management subsystem					
CVE-2022-32250	Use-after-free in Netfilter					
CVE-2022-2078	A buffer overflow in Netfilter's nft_set_desc_concat_parse() function					
CVE-2022-2602	Use-after-free in io_uring					
CVE-2021-22555	netfilter local privilege escalation					
CVE-2021-23134	Use-after-free in nfc sockets					

CVE-2021-3715	A flaw in the Traffic Control networking subsystem					
CVE-2021-33909	A variable size conversion issue on the filesystem layer					
CVE-2021-42008	Slab-Out-Of-Bounds Write vulnerability in the Linux 6pack driver					
CVE-2021-3609	Memory corruption in the CAN subsystem					
CVE-2021-3573	Use-after-free in the HCI subsystem (Blue Klotski)					
CVE-2021-27365	Heap-based buffer overflow in iSCSI driver					
CVE-2020-14381	A flaw in the futex implementation					
CVE-2020-14386	A memory corruption issue					
CVE-2019-18675	Integer overflow in cpia2					
CVE-2019-19377	Use-after-free with crafted btrfs filesystem images					
CVE-2019-19241	privilege escalation via io_uring					
CVE-2018-14634	Integer overflow in create_elf_tables()					
CVE-2018-18281	An issue with mremap()					
CVE-2018-17182	VMA use-after-free					
CVE-2017-16695	Memory corruption in BPF					
CVE-2017-11176	A flaw in the mq_notify function					
CVE-2017-1000112	UFO' privilege escalation					
CVE-2017-7308	Paclet Socket local priv esc					
CVE-2017-7184	xfrm Module Cross-Border Read-Write Escalation Vulnerability					
CVE-2017-2636	Race condition in HDLC drivers					
CVE-2017-6074	Issues with DCCP_PKT_REQUEST packet data structures					
CVE-2016-8655	af_packet					
CVE-2016-1583	Issues with the ecryptfs_privileged_open function					
CVE-2016-5195	Dirty Cow					
CVE-2016-7117	Use-after-free in the ___sys_recvmmsg function					
CVE-2015-1328	overlayfs local priv esc					
CVE-2014-4014	SGID Privilege Escalation					
CVE-2014-3153	An issue with futex_requeue function (Towelroot)					
CVE-2014-2851	Integer overflow in the ping_init_sock function					
CVE-2014-4699	ptrace/sysret vulnerability					