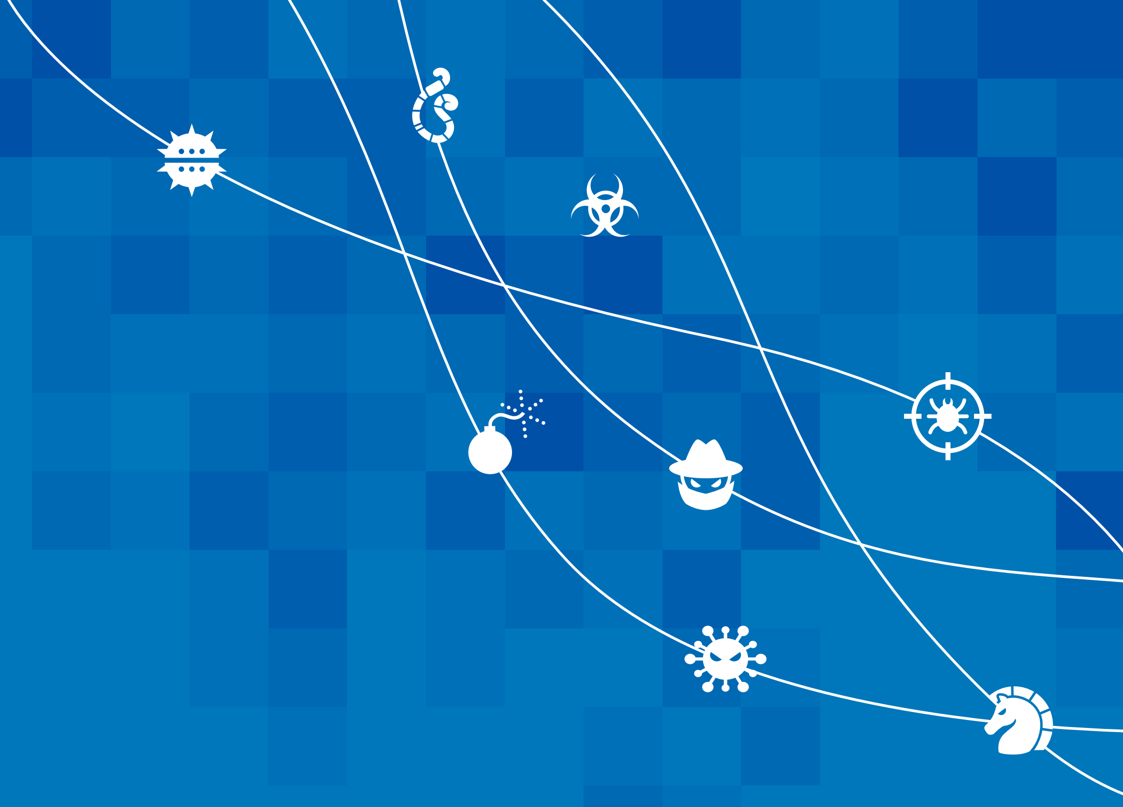


Process Control through Counterfeit Comms: Using and abusing built-in functionality to own a PLC

PREPARED BY JARED RITTLE AND PATRICK DESANTIS

Updated October 2, 2018

Process Control through Counterfeit Comms

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Programmable Logic Controllers (PLCs) are devices that factories, office buildings, and other facilities use to control the manufacturing processes running in their environment. Until recently, many PLCs have had a common problem: unauthenticated communications. These devices were designed to be reliable, but they were not built to protect against malicious actors.

Security researchers at Cisco Talos conducted an analysis of the Allen-Bradley MicroLogix 1400 PLC, resulting in the discovery of several vulnerabilities. All discovered vulnerabilities were disclosed to Rockwell Automation, the creator of the MicroLogix 1400, in accordance with Cisco's vulnerability disclosure policy on Sept. 25, 2017 and were patched in firmware version 21.004 on March 28, 2018. By leveraging a handful of these vulnerabilities, we show that it is possible to remotely program new firmware onto a MicroLogix 1400 PLC running firmware version 21.003 or below.

When a MicroLogix 1400 PLC is patched to firmware version 21.000 or above, the device is left in a state where a service critical to the firmware update process, the Simple Network Management Protocol (SNMP), is disabled and the CPU is set to a remote management mode referred to as REMOTE PROG. Before the PLC can be programmed with new firmware, it is necessary that SNMP be enabled.

We leveraged documented “Write” commands to make modifications to the system file responsible for controlling system service states (TALOS-2017-0443) on devices running firmware 21.003 and older. With SNMP fully enabled, the next step was to power cycle the device. Since there is no documented way to perform this action on a MicroLogix 1400, we searched for and discovered a way to crash the device, causing a reboot in the process (TALOS-2017-0440). Due to the state in which this crash placed the PLC, all changes made using TALOS-2017-0443 were wiped away, forcing us to search for a way to store the changes through a device power cycle. A way to store this configuration was discovered through a combination of setting the device to load its program from an installed memory module (TALOS-2017-0443) and then writing the program to that memory module (TALOS-2017-0444). With the configuration written to persistent memory, it was possible to then leverage TALOS-2017-0440 to reboot the device, leaving it in a state prepped for a firmware update.

With the necessary services enabled, we modified a valid firmware file to display “HACKED” instead of “REMOTE” when the device was placed into a specific CPU state. This can be done as long as the sum of all changes made to the firmware image equals zero, as the only error checking implemented is a checksum. The modified firmware file was then programmed to the device using its built-in update functionality, leveraging a combination of SNMP and the Trivial File Transfer Protocol (TFTP), after which the PLC successfully displayed our modified state title.

The ability to remotely program a PLC with modified firmware gives an attacker the ability to control the actions that the device performs, as well as the output that it displays. At this point, the user has to take action to protect against these kinds of attacks, as the device cannot be trusted any longer.

Users should follow these directions in order to mitigate the vulnerabilities we discovered over the course of this project. First, ensure that the MicroLogix 1400 has been updated to firmware version 21.004 or later. This firmware update fixes the vulnerabilities described in this paper.

To help harden the device against potential future vulnerabilities, the user can take specific hardening steps. Users should ensure that the device is placed into the RUN keyswitch mode when it is not being programmed. Additionally, if a memory module is installed on the device place it into Write-Protect mode, permanently preventing future remote write access to the memory module. Next, ensure that proper network segmentation is in place to prevent unauthorized systems and users from accessing control systems. Finally, enable rules 44419 - 44429 in the SNORT® Intrusion Detection and Prevention System to detect and alert on network traffic attempting to exploit these vulnerabilities.

INTRODUCTION

Process logic surrounds us in everyday life. Factories rely on it to build products, and critical infrastructure and utilities rely on it to keep necessary resources available to everyone. We rely on these processes and others like them to produce this year's favorite toy, keep us comfortable at work, and ensure that the lights stay on. But what if these processes failed? What if production of this year's toy was halted right before the Christmas rush? Or the electric company couldn't supply enough power to the neighborhood? These processes are part of what give us our quality of life, and they need to be protected.

However, PLCs, which rely on process logic, aren't necessarily secure from attackers. This was not always viewed as a problem, since the only way to communicate with a PLC was by physically connecting to its serial interface. Since then, our communication networks have expanded, and Ethernet ports have been added . With these changes, PLCs are being connected to the internet to allow for easier control for both legitimate operators and malicious actors.

As part of the "Advanced Persistent Thirst" project, security researchers at Cisco Talos conducted an analysis of the Allen-Bradley MicroLogix 1400 PLC, resulting in the discovery of hard-coded SNMP credentials. We continued our analysis of this device and discovered several additional vulnerabilities, including device fault states, unauthenticated file access, and the mishandling of invalid data. The CVEs below were patched in firmware version 21.004 on March 28, 2018. All discovered vulnerabilities were responsibly disclosed to Rockwell Automation in accordance with Cisco's vulnerability disclosure policy on Sept. 25, 2017.

- CVE-2017-12088
- CVE-2017-12089
- CVE-2017-12090
- CVE-2017-12092
- CVE-2017-12093
- CVE-2017-14462
- CVE-2017-14463
- CVE-2017-14464
- CVE-2017-14465
- CVE-2017-14466
- CVE-2017-14467
- CVE-2017-14468
- CVE-2017-14469
- CVE-2017-14470
- CVE-2017-14471
- CVE-2017-14472
- CVE-2017-14473

Throughout this report, we will walk through the methodology used while working with the MicroLogix 1400 and explain several of the vulnerabilities discovered along the way. We will show how it is possible to take a handful of vulnerabilities, some seemingly harmless, and chain them together to enable protected system services. We will then go a step further and discuss how we can leverage the newly enabled system services, along with an unpatched vulnerability to remotely program modified firmware onto a MicroLogix 1400.

TAKING OVER THE MICROLOGIX 1400

During our research, we discovered several vulnerabilities in the MicroLogix 1400 that allow a malicious actor to remotely configure, control, modify, and even disable the device. By leveraging several of these vulnerabilities in conjunction with documented and undocumented functionality, we were able to take a target device running the latest firmware at the time, starting from its post-update state and program it with modified firmware, demonstrating how dangerous a chain of seemingly harmless vulnerabilities can be, while showing the potential risks of connecting control systems to untrusted networks.

MICROLOGIX 1400

The MicroLogix line of PLCs from Rockwell Automation's Allen-Bradley are designed and built for use in a wide range of micro applications in industries such as such as printing, food and beverage processing, and waste water treatment. The MicroLogix 1400 contains a range of features and capabilities, which include an onboard input and output (I/O) module, a built-in LCD display, and remote configuration capabilities.

When PLCs are integrated into a control system, they are responsible for maintaining the physical processes conducted within that system. If a PLC is forced offline, or the device configuration is modified, the physical process that it controls can fail, potentially leading to disruption of service or destruction of equipment.

COMMUNICATIONS

Communication with the MicroLogix 1400 can be configured and controlled via its serial or Ethernet interfaces. Allen-Bradley's DF1 protocol is commonly used to conduct serial communication. When interacting with the device over its Ethernet interface, communication is supported over multiple different protocols, the most common of which is EtherNet/IP (ENIP) used to transmit Programmable Controller Communication Commands (PCCC) to and from the PLC.

ENIP is a TCP implementation of the Common Industrial Protocol (CIP) used most commonly in Allen-Bradley products. The protocol contains numerous methods of transmitting data, with the two most common being Connected and Unconnected Messages. Details regarding the protocol are publicly available within the "Communicating with RA Products using EtherNet/IP Explicit Messaging" and "EtherNet/IP Adaptation of CIP" documents.

PCCC commands are application-layer instructions transmitted to a device using any supported transmission protocol (usually DF1 or ENIP). All commands follow a structured format of command and function code pairs alongside any necessary data to trigger a specified logical operation on the device. Specification details for most of the command structure can be found in the "Allen-Bradley DF1 Protocol and Command Set" document.

DEVICE CONFIGURATION

Our device was set up to match a set of configurations similar to what may be found in the field. The latest firmware update at the time, version 21.003, was applied to the target device to ensure that it used the default configuration and contained the latest security protections. Among these security protections is the disabling of a system service necessary for programming the device firmware: the Simple Network Management Protocol (SNMP). As most devices updated to the newest firmware would not have this setting enabled, it was left disabled on the target device.

The first configuration change was made to the PLC's keyswitch mode. Most PLCs contain a keyswitch that provides physically present operators the ability to change the state of the device's CPU. By doing so, the operator can quickly control the type of actions that are authorized to be performed on the device at that point in time. It is important to note that on most PLCs, the keyswitch state can only be modified by physically interacting with the device.

On the MicroLogix 1400, three keyswitch states are possible: RUN, PROG, and REMOTE. RUN is an execution mode that allows the device to run its program, but does not allow for modification of device settings or logic. This is the recommended mode for all PLCs as it severely reduces the number of actions that can be remotely performed on the device. PROG is a programming mode that allows for updates to be made to both the device configuration and its logic. While in this mode the PLC is not running through its program. REMOTE is a programming and execution mode that blends the RUN and PROG modes.

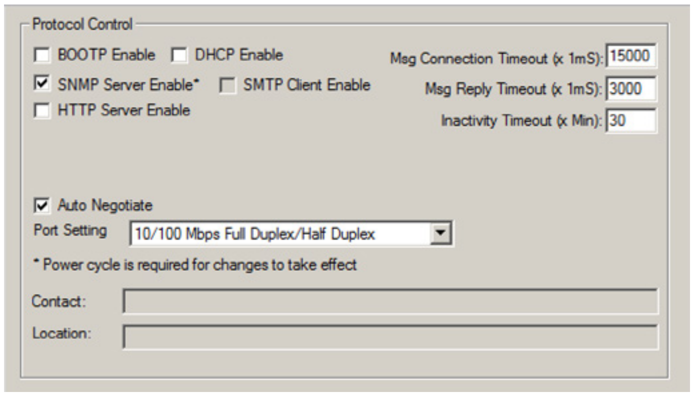
The REMOTE keyswitch mode is of particular interest, as it contains various sub-states that can be remotely changed. The two most notable of these sub-states are REMOTE RUN and REMOTE PROG. Placing the device into either of these two sub-states makes the CPU respond similarly to RUN or PROG mode while also providing additional functionality, which allows for remote state switching and communication.

The installation of an external memory module was the final modification that needed to be made to the target device before launching the attack. The MicroLogix 1400 PLC contains onboard memory that allows it to store its program, but if an error occurs that triggers the device to wipe that program, such as a corrupted download or certain fault states, the device's logic program will be lost forever. To solve this problem, the MicroLogix 1400 supports an optional memory module on which a user program can be stored.

Table 1: Device Configuration

Configuration Option	State
Device	MicroLogix 1400 Series B
CPU State	**REMOTE RUN**
SNMP	Disabled (default after FRN 21)
Memory Module	**Installed w/o Write-Protect**
ENIP	Enabled (default)
Firmware Version	21.003 or below

Figure 1: RSLogix channel configuration



Protocol Control

[ ] BOOTP Enable [ ] DHCP Enable
[✓] SNMP Server Enable* [ ] SMTP Client Enable
[ ] HTTP Server Enable

Msg Connection Timeout (x 1mS): 15000
Msg Reply Timeout (x 1mS): 3000
Inactivity Timeout (x Min): 30

[✓] Auto Negotiate
Port Setting: 10/100 Mbps Full Duplex/Half Duplex

* Power cycle is required for changes to take effect

Contact:
Location:

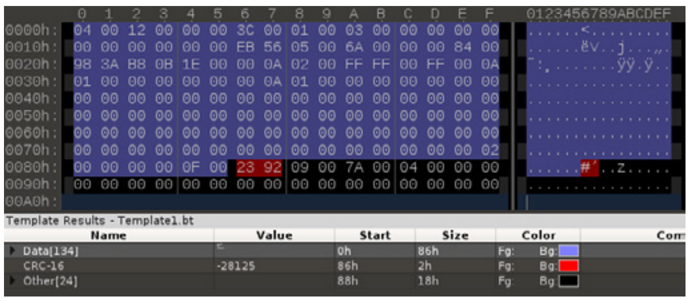
These required configuration states can be found summarized in Table 1. The settings that have been modified from their default configuration have been highlighted.

METHODOLOGY

The SNMP service must be enabled for new firmware to be programmed to the device. Prior to firmware version 21.000, SNMP was enabled by default. In version 21.000, however, the vendor chose to disable this service as a security precaution. Due to this configuration, our first target became finding a way to enable the SNMP service.

To better understand the service state modification process, we analyzed the process used by the vendor's discovery, configuration, and programming software applications — RSLinx and RSLogix — to achieve the same goal. To enable the desired service in RSLogix, a user must enter the "Channel

Figure 2: Channel Configuration File CRC

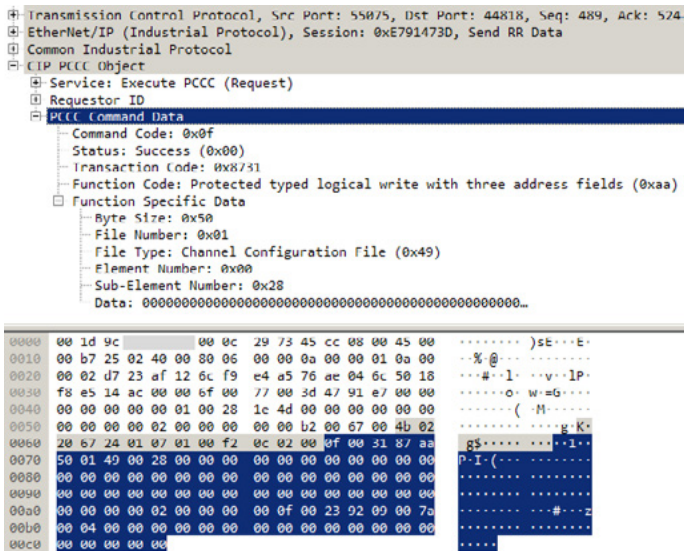


```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 04 00 12 00 00 00 3C 00 01 00 03 00 00 00 00 00   ......<.........
0010h: 00 00 00 00 00 00 EB 56 05 00 6A 00 00 00 84 00   ......ëV..j...„.
0020h: 98 3A B8 0B 1E 00 00 0A 02 00 FF FF 00 FF 00 0A   ˜:¸.......ÿÿ.ÿ..
0030h: 01 00 00 00 00 00 00 0A 01 00 00 00 00 00 00 00   ................
0040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02   ................
0080h: 00 00 00 00 0F 00 23 92 09 00 7A 00 04 00 00 00   ......#’..z.....
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00A0h:
```

Template Results - Template1.bt

Name	Value	Start	Size	Color	
Data[134]		0h	86h	Fg: Bg: 🟦	
CRC-16	-28125	86h	2h	Fg: Bg: 🟥	
Other[24]		88h	18h	Fg: Bg: ⬛	

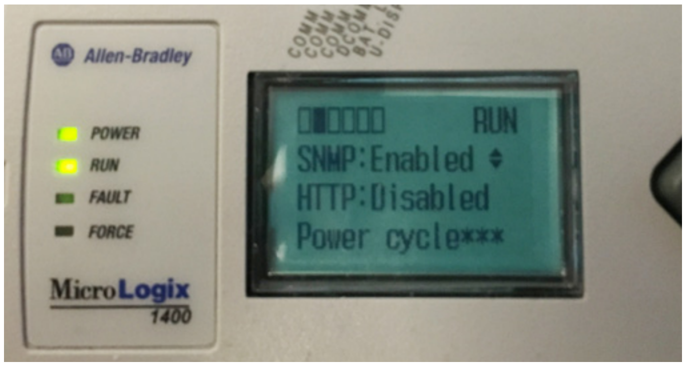
Figure 3: Example of a PCCC-protected typed logical write with three address fields command.



```
+ Transmission Control Protocol, Src Port: 55075, Dst Port: 44818, Seq: 489, Ack: 524
+ EtherNet/IP (Industrial Protocol), Session: 0xE791473D, Send RR Data
+ Common Industrial Protocol
- CIP PCCC Object
  + Service: Execute PCCC (Request)
  + Requestor ID
  - PCCC Command Data
      Command Code: 0x0f
      Status: Success (0x00)
      Transaction Code: 0x8731
      Function Code: Protected typed logical write with three address fields (0xaa)
    - Function Specific Data
        Byte Size: 0x50
        File Number: 0x01
        File Type: Channel Configuration File (0x49)
        Element Number: 0x00
        Sub-Element Number: 0x28
        Data: 0000000000000000000000000000000000000000000000000…

0000  00 1d 9c [redacted] 00 0c  29 73 45 cc 08 00 45 00   ········ )sE···E·
0010  00 b7 25 02 40 00 80 06  00 00 0a 00 00 01 0a 00   ··%·@··· ········
0020  00 02 d7 23 af 12 6c f9  e4 a5 76 ae 04 6c 50 18   ···#··l· ··v··lP·
0030  f8 e5 14 ac 00 00 6f 00  77 00 3d 47 91 e7 00 00   ······o· w·=G····
0040  00 00 00 00 00 01 00 28  1c 4d 00 00 00 00 00 00   ·······( ·M······
0050  00 00 00 00 02 00 00 00  00 00 b2 00 67 00 4b 02   ········ ····g·K·
0060  20 67 24 01 07 01 00 f2  0c 02 00 0f 00 31 87 aa    g$····· ·····1··
0070  50 01 49 00 28 00 00 00  00 00 00 00 00 00 00 00   P·I·(··· ········
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 00 00 00 00 00   ········ ········
00a0  00 00 00 00 02 00 00 00  00 0f 00 23 92 09 00 7a   ········ ···#···z
00b0  00 04 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00c0  00 00 00 00 00                                     ·····
```

Figure 4: MicroLogix 1400 SNMP state



Configuration File,” navigate to Channel 1, and mark the service as “enabled,” as shown in Figure 1. At this time, the service has not yet been fully enabled, as the device requires that a system reboot occur before the change can fully take effect.

As with many applications, there is an excessive amount of information to process when looking at this traffic in Wireshark, a network protocol analyzer. To help analyze this network traffic, we decided to filter out only the packets issued when RSLogix enables SNMP. We used RSLogix to send two legitimate Change Mode commands as markers, as there is not any built-in way to determine what this network traffic is. We were able to watch where our markers appeared in the network traffic and thereby reduce our window to a size that could be reasonably analyzed by issuing these marker commands to place the CPU into REMOTE PROG mode before enabling SNMP, and returning it to REMOTE RUN mode immediately after.

With a more manageable set of packets, we continued filtering out unnecessary commands, such as any responses from the PLC, any polling commands requesting diagnostics, and any commands issuing a “Read” of data from the device. This filtering left us with only commands issuing a “Write” operation from RSLogix to the device, but it still was too much information to properly analyze. We noticed in the network traffic that most of the packets were the same size. Thinking that, at most, a few commands should have been sent, we focused on groupings of packets with the least common length and found one that contained the device’s IP address. We knew we were on the right track since the IP address is another parameter contained within the “Channel Configuration File.”

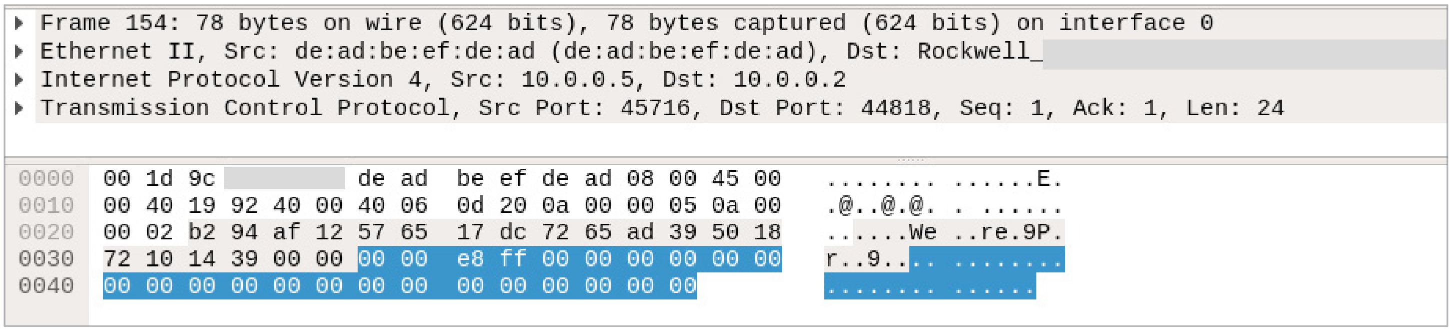
Further analysis of the network traffic revealed that there are two “Write” commands sent to the “Channel Configuration File” performed by RSLogix, each to a different part of the file. Two “Write” commands are necessary as the maximum amount of data that can be written to a file in one operation is 0x50 bytes. By comparing the combined file contents against similar runs where only system services were modified, we were able to determine that a single byte, hereafter called the “Protocol Control Byte,” controlled the state of all services.

With this information, we implemented a PCCC command similar to that used by RSLogix. When we sent this command, the PLC responded with an error code signaling that the command was invalid.

There were two additional bytes contained in the RSLogix Write network traffic that changed when modifications were made to any element of the “Channel Configuration File.” Testing revealed that a CRC-16 calculation was being performed on bytes 0x00-0x85 of the “Channel Configuration File” and placed into bytes 0x86-0x87. This breakdown can be seen in Figure 2.

We were able to successfully use a “PCCC Protected Typed

Figure 5: Device crash packet



```
▸ Frame 154: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▸ Ethernet II, Src: de:ad:be:ef:de:ad (de:ad:be:ef:de:ad), Dst: Rockwell_[redacted]
▸ Internet Protocol Version 4, Src: 10.0.0.5, Dst: 10.0.0.2
▸ Transmission Control Protocol, Src Port: 45716, Dst Port: 44818, Seq: 1, Ack: 1, Len: 24

0000  00 1d 9c [redacted] de ad  be ef de ad 08 00 45 00   ........ ......E.
0010  00 40 19 92 40 00 40 06  0d 20 0a 00 00 05 0a 00   .@..@.@. . ......
0020  00 02 b2 94 af 12 57 65  17 dc 72 65 ad 39 50 18   ......We ..re.9P.
0030  72 10 14 39 00 00 00 00  e8 ff 00 00 00 00 00 00   r..9.... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00         ........ ......
```

Logical Write with Three Address Fields" command to start the process of enabling SNMP (TALOS-2017-0443) by calculating a new CRC after updating the "Protocol Control Byte" and replacing the old CRC value. An example of this command is shown in Figure 3. When this is done, the change shown in Figure 4 will appear in the device's protocol settings.

With the "Channel Configuration File" updated, our next goal was to find a way to force the device to power cycle. The MicroLogix 1400 does not have a documented method of triggering a system reboot, so we started looking for commands where one is conducted as a result of another operation.

This search brought us to the standard firmware update process. When the firmware on a MicroLogix 1400 is being updated through the vendor's ControlFLASH software, the device reboots, enters a state informing the user that a programming operation is occurring, and reboots again to enter the normal state. We were able to capture and analyze the full details of the process.

There are three SNMP commands that get sent to the device when the device's firmware updates. The first two commands are used to configure information about the updated server, but the third is used to indicate to the device that it can start its update. If this third command is sent without either of the preceding two commands, the device will trigger its power cycle and start the firmware update process. Since there is no update server specified, the device cannot conduct the operation and restarts into the normal state (TALOS-2017-0442).

With this technique, we were successfully able to fully enable any service as long as SNMP is already enabled. If SNMP is not already enabled, however, this system reboot method cannot be conducted as it relies on that protocol.

We decided to take an approach inspired from an attempted attack on the Talos Advanced Persistent Thirst project at DerbyCon 6.0 and fuzz the device with random data. When we started piping a stream of random values across an Ethernet connection to the device on port 44818, we had little hope that anything would happen. After a few minutes, however, the device entered a fault state and rebooted. Repeated analysis and replay of the network traffic revealed a single packet that consistently triggered the device to go through this process (TALOS-2017-0440). After testing different variations of this packet, we determined that only a small portion of it is necessary. This packet is shown in Figure 5.

This power cycle method did not require SNMP to be enabled like the prior one, so it initially seemed to be a solution to our problem. Further investigation into what happens when this power cycle occurs revealed that a non-user fault gets triggered on the device. A non-user fault is a fault category caused by conditions that make the device stop execution of its logic, and often results in the user program getting erased and replaced with a default program.

When a user program gets wiped, it removes all program logic and device configuration from the device, along with any modifications we made to the "Protocol Control Byte" with it. To get around this problem, we started looking into how the device loads its default program.

Investigation into the location of this default program revealed that it is not stored in a location that can be easily modified. It also revealed an option to instruct the device to load its program from an installed memory module when an error is encountered instead of loading its default program (TALOS-2017-0443).

By installing a memory module and instructing the device to load from that memory module on error via a PCCC command wrapped in the device's "Programming Routine" (See Appendix D), we were able to get the online user program into the desired pre-reboot state.

Figure 6: Enable SNMP full attack path

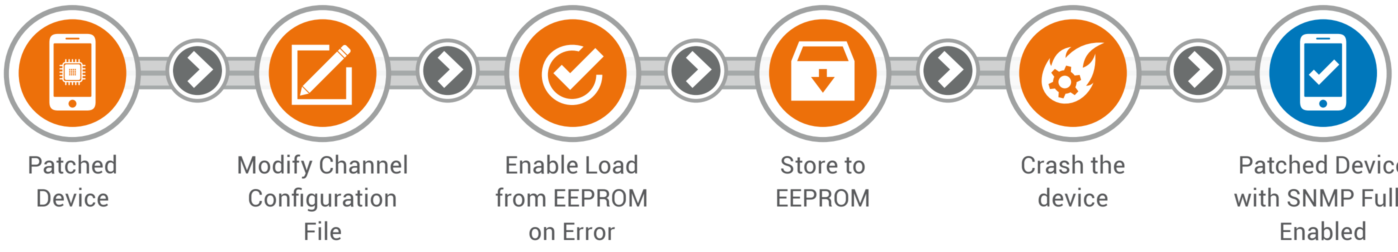
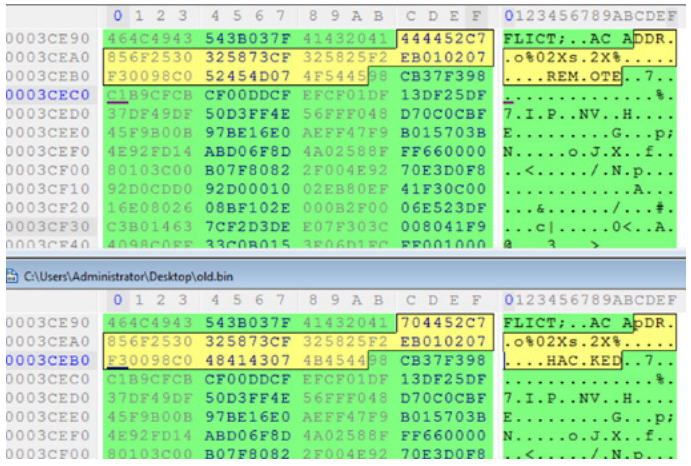

Patched Device
Modify Channel Configuration File
Enable Load from EEPROM on Error
Store to EEPROM
Crash the device
Patched Device with SNMP Fully Enabled

Figure 7: Modifying the firmware

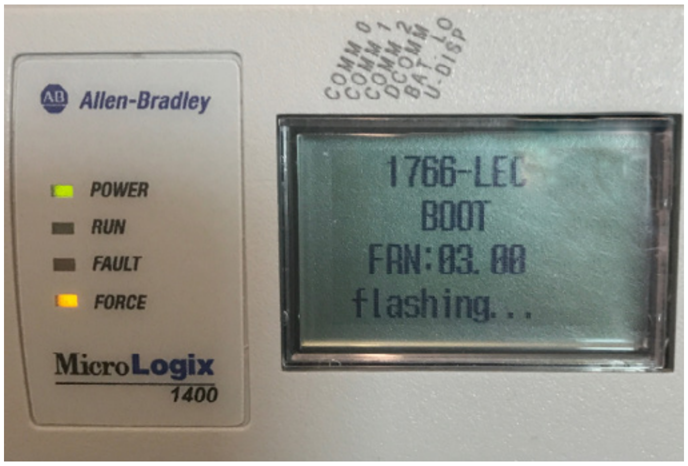


```
          0 1 2 3  4 5 6 7  8 9 A B  C D E F  0123456789ABCDEF
0003CE90  464C4943 543B037F 41432041 444452C7 FLICT;..AC ADDR.
0003CEA0  856F2530 325873CF 325825F2 EB010207 .o%02Xs.2X%.....
0003CEB0  F30098C0 52454D07 4F544598 CB37F398 ....REM.OTE..7..
0003CEC0  C1B9CFCB CF00DDCF EFCF01DF 13DF25DF ..............%.
0003CED0  37DF49DF 50D3FF4E 56FFF048 D70C0CBF 7.I.P..NV..H....
0003CEE0  45F9B00B 97BE16E0 AEFF47F9 B015703B E.........G...p;
0003CEF0  4E92FD14 ABD06F8D 4A02588F FF660000 N.....o.J.X..f..
0003CF00  80103C00 B07F8082 2F004E92 70E3D0F8 ..<...../.N.p...
0003CF10  92D0CDD0 92D00010 02EB80EF 41F30C00 ............A...
0003CF20  16E08026 08BF102E 000B2F00 06E523DF ...&....../...#.
0003CF30  C3B01463 7CF2D3DE E07F303C 008041F9 ...c|.....0<..A.
0003CF40  4098C0FF 33C0B015 3F06D1FC FF001000 @   3   >
```

C:\Users\Administrator\Desktop\old.bin

```
          0 1 2 3  4 5 6 7  8 9 A B  C D E F  0123456789ABCDEF
0003CE90  464C4943 543B037F 41432041 704452C7 FLICT;..AC ApDR.
0003CEA0  856F2530 325873CF 325825F2 EB010207 .o%02Xs.2X%.....
0003CEB0  F30098C0 48414307 4B454498 CB37F398 ....HAC.KED..7..
0003CEC0  C1B9CFCB CF00DDCF EFCF01DF 13DF25DF ..............%.
0003CED0  37DF49DF 50D3FF4E 56FFF048 D70C0CBF 7.I.P..NV..H....
0003CEE0  45F9B00B 97BE16E0 AEFF47F9 B015703B E.........G...p;
0003CEF0  4E92FD14 ABD06F8D 4A02588F FF660000 N.....o.J.X..f..
0003CF00  80103C00 B07F8082 2F004E92 70E3D0F8 ..<...../.N.p...
```

Figure 8: Flashing modified firmware



With the user program ready, the final necessary element was finding a way to write that program to the memory module. We knew it was possible since this functionality was implemented with RSLogix. Using Wireshark, we captured and analyzed the network traffic, resulting in the discovery of a command that will be hereafter referred to as Store to EEPROM (TALOS-2017-0444). We could now combine everything that we learned to fully enable SNMP on our target device using this method.

A summarized view of the path that we took to fully enable SNMP is shown in Figure 6.

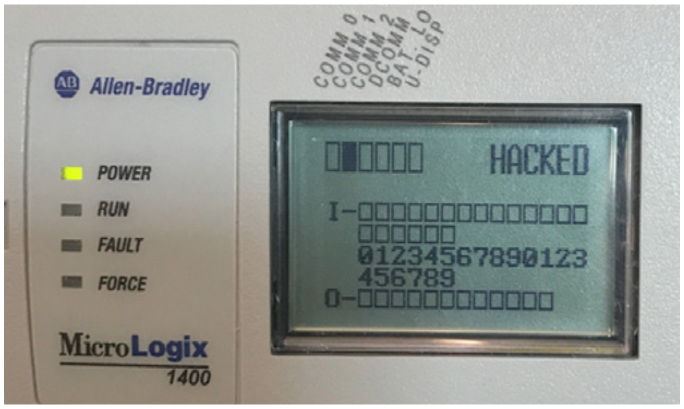
With the initial goal of fully enabling SNMP on the target device accomplished, the next step was to update the firmware. To do this, we used a technique described in the DEFCON25 talk "From Box to Backdoor: Using Old School Tools and Techniques to Discover Backdoors in Modern Devices" by Patrick DeSantis.

We needed to build a firmware image before the device could be programmed. To do this, we started with a copy of the newest firmware at the time. In the MicroLogix 1400 Series B, the only check in place to ensure that a firmware file is valid is a basic checksum. Since only checksum validation is used, it is possible to make modifications to the firmware file as long as the net sum of any changes made is zero. Using this strategy, we created a modified copy of firmware version 21.003 where the LCD CPU state indicator of "REMOTE" was changed to read "HACKED", making sure to offset our changes in other nonessential surrounding text. An example of the modifications can be seen in Figure 7.

With the modified firmware in hand, it was time to program it onto the device. The firmware update process on a MicroLogix 1400 contains two main phases: a setup phase and a programming phase. During the setup phase, three SNMP commands are issued to the device. These commands configure the device with an IP address from which to load the new firmware, specify the name of the firmware file to load, and trigger a power cycle. If done correctly, the device will reboot and begin transferring the modified firmware from the specified server using the Trivial File Transfer Protocol (TFTP). Once the firmware file is fully downloaded, the device will proceed with programming the modified firmware, as shown in Figure 8.

When the device completes the programming process, it triggers another power cycle and boots into the standard operating mode. At this stage, due to the specific changes we made it was visually apparent that the device was running modified firmware. From this point forward, until the device is reprogrammed, any time the device is placed into the REMOTE keyswitch state the LCD displays "HACKED" instead. This can be seen in Figure 9.

Figure 9: Modified Firmware



IMPACT

While this example demonstrates the modification of a display element on the device, it is important to remember that the modification was made at the firmware level. Using the same technique that we used to modify the display it is possible to make any desired change to the firmware as long as the checksum remains the same. With enough work, a malicious actor can use this ability to build valid firmware with modifications, ranging from minor display updates like the one shown above, to major changes of the control process.

An attacker has the ability to make the PLC do whatever they want once they have control over its firmware. This access allows them to make any desired modification, including but not limited to, inserting backdoors for future access or making the device report false information to its operators to report incorrect states.

Once this type of access has been gained by an attacker, the device and every status that it reports can no longer be trusted.

MITIGATION

The easiest and quickest way to prevent most of the attacks discussed in this paper is to follow proper network segmentation procedures, ensuring that access to the MicroLogix 1400 and any other control systems is restricted to only necessary systems and users.

Make sure that the most up-to-date firmware is being used. At time of writing, that is version 21.004.

If possible, ensure that the PLC is placed into the RUN keyswitch mode whenever it is not being programmed, regardless of the firmware version.

If a memory module is in use, make sure to use the "Write-Protect" feature after writing the desired program. Doing so will permanently disable any future logical writes to the memory module, preventing an attacker from leveraging TALOS-2017-0444 and making it more difficult to enable sensitive services.

It is important to note that by enabling the Write-Protect feature, it will not be possible to make any legitimate changes to the program stored on that module after it has been enabled.

Lastly, monitor your network for any suspicious network traffic with an Intrusion Detection System such as Snort. The Snort rules listed below in the "Snort Coverage" section below can be used to alert users to many of the vulnerabilities discussed in this article.

SNORT COVERAGE

The following Snort rules can aid in the detection of the vulnerabilities described in this report:

- 44419
- 44420
- 44421
- 44422
- 44423
- 44424
- 44425
- 44426
- 44427
- 44428
- 44429

REFERENCES

- Allen-Bradley Rockwell Automation. (1996, October). DF1 Protocol and Command Set. Retrieved from Rockwell Automation Literature Library: literature.rockwellautomation.com/idc/groups/literature/documents/rm/1770-rm516_-en-p.pdf
- Allen-Bradley Rockwell Automation. (2001, June). Communicating wtih RA Products using EtherNet/IP Explicit Messaging. Retrieved from Rockwell Automation Downloads: rockwellautomation.com/resources/downloads/rockwellautomation/pdf/sales-partners/technology-licensing/eipexp1_2.pdf
- Allen-Bradley Rockwell Automation. (2017, March). MicroLogix 1400 Programmable Controllers User Manual. Retrieved from Rockwell Automation Literature Library: literature.rockwellautomation.com/idc/groups/literature/documents/um/1766-um001_-en-p.pdf
- Buvel, R. (n.d.). python crcmod 1.7. Retrieved from Python Software Foundation Package Index: pypi.python.org/pypi/crcmod
- Carnegie Mellon University. (2010, January). CVE-2009-3739. Retrieved from Vulnerability Notes Database: https://www.kb.cert.org/vuls/id/144233
- Cisco. (2016, November). Vendor Vulnerability Reporting and Disclosure Policy. Retrieved from Cisco Security Center: https://www.cisco.com/c/en/us/about/security-center/vendor-vulnerability-policy.html
- Cisco Talos. (2016, August). TALOS-2016-0184: AB Rockwell Automation MicroLogix 1400 Code Execution Vulnerability. Retrieved from Talos Vulnerability Reports: https://www.talosintelligence.com/reports/TALOS-2016-0184
- ControlNet International and Open DeviceNet Vendor Association. (2001, June). EtherNet/IP Adaptation of CIP Specification. Retrieved from http://read.pudn.com/downloads166/ebook/763212/EIP-CIP-V2-1.0.pdf
- DeSantis, P. (2017). From Box to Backdoor: Using Old School Tools and Techniques to Discover Backdoors in Modern Devices. Retrieved from Defcon Media Server: FROM BOX TO BACKDOOR Using Old School Tools and Techniques to Discover Backdoors in Modern Devices
- Liechti, C. (n.d.). pySerial. Retrieved from pySerial 3.0 Documentation: pythonhosted.org/pyserial
- MITRE. (2012, August). CVE-2012-4690. Retrieved from Common Vulnerabilities and Exposures: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4690
- MITRE. (2018, January). CVE-2017-16740. Retrieved from Common Vulnerabilities and Exposures: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16740
- plctalk.net. (2008, February). PLCTalk Message Board. Retrieved from plctalk.net/qanda/showthread.php?t=37220
- Rockwell Automation Allen-Bradley. (n.d.). MicroLogix 1400 Programmable Logic Controller Systems. Retrieved from Rockwell Automation Software: https://ab.rockwellautomation.com/Programmable-Controllers/MicroLogix-1400#applications
- Santamarta, R., & Wightman, K. (n.d.). multi_cip_command - Metasploit Framework. Retrieved from Github: https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/admin/scada/multi_cip_command.rb
- Tacliad, F., Nguyen, T. D., & Gondree, M. (2017, December). DoS Exploitation of Allen-Bradley's Legacy Protocol through Fuzz Testing. Retrieved from https://www.gondree.com/pdfs/tacliad_icss_17.pdf

APPENDIX A - PCCC CMD/FNC CODE QUICK REFERENCE

Command and function codes referenced in Table 2 below were extracted from the Allen-Bradley "DF1 Protocol and Command Set" document, as well as online resources.

Table 2: Relevant PCCC CMD/FNC Code

CMD	FNC	Description
0x06	0x03	Request Diagnostics
0x0F	0x11	Get Edit Resource
0x0F	0x12	Return Edit Resource
0x0F	0x52	Download Complete
0x0F	0x58	Store to EEPROM
0x0F	0x80	Change Mode
0x0F	0x88	Execute Command List
0x0F	0x8F	Apply Port Configuration
0x0F	0xA2	Protected Typed Logical Read with Three Address Fields
0x0F	0xAA	Protected Typed Logical Write with Three Address Fields

APPENDIX B - PCCC FILE TYPE QUICK REFERENCE

File type and number mappings referenced in Table 3 were pulled from a combination of the Allen-Bradley "DF1 Protocol and Command Set" document, as well as our own analysis of communications.

Table 3: PCCC File Type Quick Reference

File Name	File Type	Common File Number
Unknown, but useful	0x00	0x00
Unknown, but useful	0x01	0x00
Unknown, but useful	0x02	0x00
Unknown, but useful	0x03	0x00
Ladder Logic File	0x22	0x02
Function File - CS0 & CS2	0x48	0x00
Channel Configuration File	0x49	0x01
Function File - ES1	0x4A	0x01
Online Edit File	0x65	0x00
Function File - IOS	0x6A	0x00
Data File - OUTPUT	0x82	0x00
Data File - INPUT	0x83	0x01
Data File - STATUS	0x84	0x02
Data File - BINARY	0x85	0x03
Data File - TIMER	0x86	0x04
Data File - COUNTER	0x87	0x05
Data File - CONTROL	0x88	0x06
Data File - INTEGER	0x89	0x07
Data File - FLOAT	0x8A	0x08
Force File - OUTPUT	0xA1	0x00
Force File - INPUT	0xA2	0x01
Function File - ES0	0xE0	0x00
Function File - STI	0xE2	0x03
Function File - EII	0xE3	0x00
Function File - RTC	0xE4	0x00
Function File - BHI	0xE5	0x00
Function File - MMI	0xE6	0x00
Function File - LCD	0xEC	0x00
Function File - PTOX	0xED	0x00
Function File - PWMX	0xEE	0x00

APPENDIX C - PROTOCOL CONTROL BYTE BITFIELD

When attempting to change the state of a service on the MicroLogix 1400, an update to the “Channel Configuration File” is required. Within the data section of this file there is one byte, referred to as the “Protocol Control Byte” that determines the state of each of the supported services. This byte served as a bitfield, with each bit representing the state of a particular service. Table 4 provides a breakdown of the bit to service mapping that is implemented in the MicroLogix 1400 PLC.

Table 4: Protocol Control Byte

7	6	5	4	3	2	**1**	0
ENIP	Unknown	ModbusTCP	DNP3	Unknown	SMTP	**SNMP**	HTTP

APPENDIX D - PROGRAMMING ROUTINE

Table 5: Execute Command List

CMD	STS	TNS	FNC	Number of FNCs	FNC 1 Length	FNC1	FNC1 Data	FNC 2 Length	FNC 2
0F	00	- -	88	02	0C	AA	See breakout below	01	56

Byte Size	File Number	File Type	Element Number	Sub Element Number	Address 1	Address 2	Address 3
06	00	63	00	00	0C 91	00 00	83 F1

When attempting to perform certain commands, an error is encountered that indicates that the necessary privileges were not met. Investigation uncovered that this error comes from not having supplied the proper sequence of instructions before and after attempted execution of a desired command. This sequence will hereafter be referred to as the "Programming Routine." To successfully go through the "Programming Routine," the device must be in the REMOTE keyswitch mode.

To initiate a "Programming Routine," the following three setup commands must be successfully executed: "Change Mode - REMOTE PROG," "Execute Command List," and "Get Edit Resource."

Once all three setup commands have been successfully executed, the desired commands can be run. There does not appear to be any limit on the number of commands sent as long as they are all within the same session.

After all desired commands have been successfully received, the following commands must be executed to ensure that all changes are saved: "Download Completed," "Apply Port Configuration," "Return Edit Resource" and, optionally, "Change Mode - REMOTE RUN."

All of these commands are well documented within the "DF1 Protocol Command Set" manual, except for "Execute Command List." A potential breakout and example packet for the "Execute Command List" command is shown in Table 5. This information was obtained from discussion on a PLCTalk forum, as well as our own internal analysis.

When properly executed, the structure of the "Programming Routine" should follow the order laid out below.

1. Change Mode - REMOTE PROG
2. Execute Command List
3. Get Edit Resource
4. Any Commands That Require the Routine
5. Download Completed
6. Apply Port Configuration
7. Return Edit Resource
8. Change Mode - REMOTE RUN (optional)

If no errors are encountered along the way, the desired changes should be reflected on the device. It is important to note that this method is not required for all commands. Even commands that do not require it can still be run within the routine.