Leveraging Ethernet Card Vulnerabilities in Field Devices

Daniel Peck, Dale Peterson Digital Bond, Inc. Fort Lauderdale, Florida {peck, peterson}@digitalbond.com

Abstract: Field devices essential for the monitoring and control in DCS and SCADA systems are increasingly being deployed with Ethernet cards to connect these devices to local and wide area IP networks. Many of the Ethernet cards have their own CPU, memory, operating system and applications. Field device vendors are also providing the capability to upgrade or replace the firmware in these Ethernet cards. Unfortunately in most cases there is no effective security on the firmware upload to the field device Ethernet cards.

In this paper the authors demonstrate how using commonly available tools an attacker can learn how firmware is loaded into two different field device Ethernet cards, write his own malicious firmware, and load that malicious firmware into the field device Ethernet cards. After this proof of concept malicious firmware load, the authors discuss different classes of attacks that could be launched against the process being controlled, other field devices and other systems on the control system network.

The paper concludes with a brief discussion and examples of vulnerabilities in common management services, such as HTTP, FTP and SNMP, found on many field device Ethernet cards.

Keywords: Unauthenticated Firmware Upload, PAC, PLC, RTU, Controller, Field Device, Field Device Worm, Field Device Ethernet Card

1 Introduction

Field devices are critical components of control systems. They communicate with sensors and actuators to monitor and control a process. The increasing computing power and capabilities in field devices allows asset owners to push more intelligence to these field devices, and they become much more than communication devices between an HMI or control center and the sensors and actuators on the plant floor or field site. Common terms for field devices include Programmable Logic Controllers (PLC), Programmable Automation Controllers (PAC), Remote Terminal Units (RTU) and Intelligent Electronic Devices (IED).

As with most control system components, field devices were designed for a closed, trusted network with little thought for security. If an attacker can ping the typical field device, and has a client or HMI to communicate with the field device, he can send any

read, write, diagnostic or configuration command to the field device. This is due to a lack of source or data authentication in most control system protocols. Some examples of control system protocol based attacks include:

- Writing new values to set points, thereby changing the process
- Rebooting the field device continuously to affect availability
- Setting the field device to listen only mode so it will not respond to requests

Digital Bond has developed network Intrusion Detection System (IDS) signatures that have been integrated into most commercial IDS sensors to identify these attacks for Modbus TCP, DNP3, and ICCP [1].

A similar lack of security features is often found in non-control system protocols such as Telnet, FTP, HTTP and SNMP that are used to manage a field device. Some field devices support one or more user accounts for login, but even this feature is rarely used.

Beyond demonstrating the ease in exploiting an insecure control system or management protocol, most field device security analysis and testing to date has involved fuzz testing the field device's protocol stack [2]. Unfortunately many field devices have failed even minor fuzz testing of Ethernet, IP and TCP/UDP protocols. Some have even failed when legitimate, properly formatted broadcast and multi-cast packets are sent to the field device. Failure most often is a crash of the Ethernet interface or the entire field device and requires a reboot to resume normal operations.

Given this extremely vulnerable state of most field devices, why is further testing or analysis of exploits warranted? For at least two reasons:

- 1. A knowledgeable attacker could stage an attack for a time and purpose with maximum impact, and this could be done well after the compromise. This paper will discuss some potentially devastating scenarios.
- 2. A compromised field device can be used as an attack platform for other elements in the network. There are tens or hundreds of field devices in control systems that could be used for individual or coordinated attacks. Imagine a distributed denial of service (DDOS) from the field devices on all servers and workstations in the control center. A field device is also less likely to be analyzed for its security posture as compared to an HMI running Windows or a server running Unix. This is analogous to the use of compromised printers as an attack platform in a corporate network.

The easiest point of field device access for a cyber attacker is the Ethernet card. Many of these cards have their own CPU, RAM and operating system. The computers on these cards can be attacked and often exploited. Once exploited, they can be used to attack the CPU and other cards on the field device as well as other devices on the control system LAN and WAN.

This paper describes different classes of attacks on field device Ethernet cards, sample vulnerabilities found in numerous field devices and corresponding exploits, and the potential consequences of these exploits assuming a knowledgeable attacker.

2 Rogue Firmware Load

Many field devices allow the firmware to be updated over the network. This is also often true of field device Ethernet cards with their own computing resources. A remote firmware update can be very useful, especially in a geographically dispersed SCADA network, for recovery from system corruption, patching newly discovered problems or updating the field device software with new features. It also can be very useful for an attacker if it is not properly authenticating the source and data being uploaded prior to accepting and loading this new firmware.

In the multiple field devices that Digital Bond has in its lab, not one of these field devices authenticates the source or data prior to accepting and loading the firmware. If it is possible for a legitimate user to upload new firmware over the network, it is possible for an attacker, with some work at the assembly code level and logical access to the field device, to upload his own firmware.

In Section 2, the authors will describe the steps required to load arbitrary firmware on two different systems and some possible ways an attacker might use this capability beyond simply attacking the selected field device. It is important to note that this problem is widespread, and goes much beyond these two systems. Any vendor or asset owner that has field devices that allow unauthenticated or weak authentication firmware uploads should be concerned and contact their vendor for guidance and a solution.

2.1 Systems Described

Our test devices for this research were the Rockwell 1756 ENBT Ethernet module and the Koyo H4-ECOM100 Ethernet module, see Figure 1. While similar in functionality these devices were built on very different platforms.

The 1756 ENBT module is a PowerPC device running a VxWorks 5.5 operating system, intended to be managed primarily by the RSLogix family of products. Services available on the device include EtherNet/IP, HTTP, and SNMP. FTP is also available with a little more work, as we'll discuss later.

The H4-ECOM100 module is a device based on a Nios Altera Field Programmable Gate Array (FPGA) chipset. The services available on this device include HTTP and MODBUS/TCP.

While the web interface on the 1756 ENBT is intended almost exclusively for displaying diagnostic information, the H4-ECOM100 web server is used extensively for configuration and maintenance activities.



Figure 1 – Koyo H4-ECOM100 and Rockwell Automation 1756 ENBT Modules

2.2 Learning and Exploiting the Load

We began the process by obtaining several sets of firmware images from the Rockwell vendor site. Several versions were downloaded to allow for comparison, to help in identifying static fields, and various "special" locations in the firmware. Our first step in the analysis was to examine the binaries themselves. In this analysis we find several different segments, some with readable text, that appear to be some of the web pages that the device serves. These appear to exist in a CISF FTL partition, which is similar enough to FAT16. These will be investigated later, but for now we will move on and continue looking for the actual code that gets executed, and being that we are at a bit of a dead end here we'll start to rely of assumptions and experience.

lex Worksl File Edit				ow Help												
: 🖬 🖬 :	- 1	h f		8 8		e ei	0 B	S L	0 F	DF	i en lli				M .	
	a a				40		9 9	9 1	ų r	n lG	9 🖭	<u> </u>			<u> </u>	
· ~ <<	» 또 .	<u>2 2 2 2</u>	≌^ ∣	-3	* +		1 %	[<] [>]	A† a	u↓ a∯A	P	88	% 🖤	Ē		
											11					
0190302								FFFF								
0190310	ODOA	093C	6865	6164	3E0D			6C69			656C				. <head><link href="/css/radev</td><td></td></tr><tr><td>0190350</td><td></td><td>652E</td><td></td><td></td><td></td><td></td><td></td><td>2274</td><td></td><td></td><td></td><td></td><td></td><td>ic</td><td>e.css" rel="st</td><td></td></tr><tr><td>0190336</td><td>5 7960</td><td>6573</td><td>6865</td><td>6574</td><td></td><td></td><td></td><td>3D22</td><td></td><td></td><td></td><td></td><td></td><td> y1</td><td>esheet" type="text/css"/></head>	
0190367	A ODOA	0909	3C73	6372	6970	7420	7479	7065	3D22	7465	7874	2F6A	6176		<script language="JavaScr</td><td></td></tr><tr><td>019039E</td><td>5 6970</td><td>7422</td><td>2073</td><td>7263</td><td>3D22</td><td>2F73</td><td>6372</td><td>6970</td><td>7473</td><td>2F55</td><td>524C</td><td>6861</td><td>6E64</td><td>lip</td><td>t" src="/scripts/URLhand</td><td></td></tr><tr><td>01903B8</td><td>3 6065</td><td>2E6A</td><td>7322</td><td>3E3C</td><td>2F73</td><td>6372</td><td>6970</td><td>743E</td><td>ODOA</td><td>0909</td><td>ODOA</td><td>0909</td><td>3C73</td><td>lle</td><td>.js" type="text/jav</td><td></td></tr><tr><td>0190384</td><td>4 6173</td><td>6372</td><td>6970</td><td>7422</td><td>206C</td><td>616E</td><td>6775</td><td>6167</td><td>653D</td><td>224A</td><td>6176</td><td>6153</td><td>6372</td><td>as</td><td>cript"></script>	
01903D2											7BOD				le>link{	
01903E0	10963	6F6C		3A20				633B							olor: #0066cc;text	
0190406		6563			696F			6E64							ecoration: underline:	
0190420		0963													.cursor: hand;}	
0190420								0A09							indicator{f	
0190454		742D						6076							t-family: helvetica;	
0190454 019046E				2D73									636F			
								3270							font-size: 12px;co	
0190488		723A													r: #333333;font-we	
01904A2		6874		626F				0909					3A20		ht: bold;display:	
01904B0								0A09							ne;}.	
01904D6				6865				0A3C							<body bgcol<="" td=""><td></td></body>	
01904F0		3D22													="#b7b6b6" onload="frame	
0190502		6865		2829				3C64						_cl	heck();"> <div style="</td><td></td></tr><tr><td>0190524</td><td>4 6D61</td><td>7267</td><td>696E</td><td>2D6C</td><td></td><td></td><td></td><td>3070</td><td></td><td></td><td></td><td></td><td></td><td></td><td>rgin-left: 10px;"><!---</td--><td>-</td></div>	-
019053E		5374						2D3E							Start tabs> <table< td=""><td></td></table<>	
0190558	3 7769	6474	683D	2239	3825	2220	626F	7264	6572	3D22	3022	2063	656C	Wi	dth="98%" border="0" cel	
0190572	2 6073	7061	6369	6E67	3D22	3022	2063	656C	6C70	6164	6469	6E67	3D22	15	pacing="0" cellpadding="	
99472767.	wbn	PN-19870														
ffset: 1639:	180 [0×00	19030C]								Fir	id Results			_		(Î + Î)
8BIT Signed B	Byte	1									ddress				Length	
8BIT Unsigne		255														
6BIT Signed S		1														
SBIT Unsigne																
2BIT Signed L		1														
28IT Unsigne			95						_							
KBIT Signed (1	22200EE	1615												
2811 Float		1.#ONAN		1012												
Stati Picat		1.#QNAN							_		-					
UDITINATE	_	0.00.00.	un resigns	1000					-	1						
Data Inspe	ctor / S	ructure Vie	wer /							11/2	Compare /	\ Checks	um ∖Fin	id (I	Bookmarks 👌 Output /	

Figure 2 - HTML Visible in Firmware

Using a tool developed by Matasano named deezee, we scan the firmware image looking for zlib compressed sections, and we get a hit. Pulling this into our hex editor and examine some strings, we find what appears to be some symbol names, such as AB_Exit at 0x001ef0b4 and AB_Init at 0x001ef0ac. These seem interesting but we have to find the offset table to be able to do anything very useful. Looking through the file we notice a very regular ten-byte pattern beginning at 0x001fbdf8.

ex Workshop - [99472767.wbn.0] File Edit Disk Options Tools Window Help		
🗃 🖬 🖶 🎗 🛍 🖻 그 오 🔌 🌾 🛍 🔚 🗑 🖬 🖇 L Q F I		
~ << >> << >> << >> << >> << >< << >< << >< << >< << >< << >< << >< << >< << >< << >< << <	* th ff ff % # E	
D1FBDEA BDAC 0000 0000 0000 0000 0000 0000 002E F0B4 0		<u> </u>
D1FBE04 0000 0000 002E F0AC 0011 959C 0000 0500 0000 0		
D1FBE1E 9608 0000 0500 0000 0000 002E F098 0034 B26C 0		
D1FBE38 002E F084 001C 4740 0000 0500 0000 0000 002E F		
D1FBE52 0500 0000 0000 002E F050 001C 4A90 0000 0500 0		
	B64 0000 0500 0000d	
D1FBE86 0000 002E FOOC 001C 8C40 0000 0500 0000 0000 0		
D1FBEA0 0000 0500 0000 0000 002E EFE8 001C 8E70 0000 0		
	01C 8E80 0000 0500	
	000 002E EFA0 001CKK	
	1000 0500 0000 0000	
D1FBF08 002E EF80 001C 8F04 0000 0500 0000 0000 002E B		
	1000 0000 002E EF50dX@P	
	084 0000 0500 0000	
D1FBF56 0000 002E EF2C 001C 9120 0000 0500 0000 0000 0	02E EF18 001C 5D70	
D1FBF70 0000 0500 0000 0000 002E EF08 001C 5ED4 0000 (500 0000 0000 002E^	
D1FBF8A EEFO 001C 91E8 0000 0500 0000 0000 002E EEDC 0	01C 6070 0000 0500	
D1FBFA4 0000 0000 002E EECC 001C 61D4 0000 0500 0000 0	000 002E EEC4 0033aa	
D1FBFBE 8D18 0000 0900 0000 0000 002E EEB4 001D 1648 (000 0500 0000 0000HH.	
D1FBFD8 002E EEA0 001D 1CFC 0000 0500 0000 0000 002E B	E88 001D 1BCO 0000	
D1FBFF2 0500 0000 0000 002E EE74 001D 1550 0000 0500 0	000 0000 002E EE60tP	
D1FC00C 001D 16C0 0000 0500 0000 0000 002E EE50 001D 1	OFO 0000 0500 0000PP	
D1FC026 0000 002E EE3C 001D 1A8C 0000 0500 0000 0000 0	02E EE24 001D 1784<	_
D1FC040 0000 0500 0000 0000 002E EE10 001D 1B00 0000 0	500 0000 0000 002E	
D1FC05A EDFC 001D 130C 0000 0500 0000 0000 002E EDE4 0		-
) 99472767 🕅 99472767.w	· · · · · · · · · · · · · · · · · · ·	
	× 4185 instances of '00000500' found in 99472767.wbn.0	
ffset: 2080248 [0x001FBDF8]		
8817 Signed Byte 0	Address Length	
881 Unsigned Byte 0	001D7F85 00000004	
⁸⁸¹ Signed Short 46	001D8019 00000004	
६८१७ Unsigned Short 46 १८१७ Signed Long 3076276	001D84E9 00000004	
28/1 Signed Long 3076276 28/1 Unsigned Long 3076276	001D84ED 00000004	
VBIT Signed Quad 13212504814622072	00108525 00000004	
Kin Signed Quad 13212504814622072 Isigned Quad 13212504814622072	001D8529 00000004 001D852D 00000004	
28/ Float 4.3107808e-039	00108520 0000004	
4817 Double 8.6055417e-308		Þ
Data Inspector & Structure Viewer /	Compare A Checksum Find Bookmarks Output	<u> </u>
Data inspector A Dedetate viewer		

Figure 3 - Ten Byte Pattern: 0x 0000 0500 0000 0000 002E

The first address looks a lot like the address of AB_Exit, and the second one looks a lot like the address of AB_Init. Some quick subtraction and we find that our load address is 0x00100000, and it looks like the next entry in that repeating pattern is the address of the code itself. We're almost ready to begin some real analysis, but first we have to know what kind of processor this code was meant to run on, our first guess was that it was on an ARM chip, but looking at the firmware dump we see r0, sp, r2, and quite a few others, we have our answer, a Power PC processor.

Hex Workshop - [99472767.wbm.0] File Edit Disk Options Tools Window Help		_ 0 2
ᆃᆲᇣᆇᆝᆠᄩᄤᇐᇎᆙᇂᢦᆙᄜᆝᆙᇔᄬᆝᇥᆙႽᆝᆝᆝᅣ᠐		
≒~ ≪ ≫ 쓰 꼰 쓴 꼭 ^ & ᡟ- + - ★ / % 区 ▷ A↑ a↓·	9 B B B 🗱 🗱 🛢 🗌	
001DAA22 0000 0000 0000 0000 0000 0000 0000	000 0000 6672 3330fr31fr30	-
001DAA3C 0000 0000 6672 3239 0000 0000 6672 3238 0000 0	000 6672 3237 0000fr29fr28fr27	_
001DAA56 0000 6672 3236 0000 0000 6672 3235 0000 0000 6	72 3234 0000 0000fr26fr25fr24	
001DAA70 6672 3233 0000 0000 6672 3232 0000 0000 6672 3	231 0000 0000 6672 fr23fr22fr21fr	
001DAA8A 3230 0000 0000 6672 3139 0000 0000 6672 3138 0	000 0000 6672 3137 20fr19fr18fr17	
	000 6672 3134 0000fr16fr15fr14	
	72 3131 0000 0000 fr13 fr12 fr11	
	700 6672 3600 6672 fr10fr9.fr8.fr7.fr6.fr	
001DAAF2 3500 6672 3400 6672 3300 6672 3200 6672 3100 6		
001DABOC 7200 0000 7865 7200 6372 0000 7063 0000 6374 7		
	232 3700 7232 3600 r.r31.r30.r29.r28.r27.r26.	
001DAB40 7232 3500 7232 3400 7232 3300 7232 3200 7232 3		
001DAB5A 3900 7231 3800 7231 3700 7231 3600 7231 3500 7		
001DABJA 7231 3200 7231 3100 7231 3000 7239 0000 7238 0		
	370 0000 7230 0000r5r4r3r2spr0	
DO1DABA8 7A65 726F 696E 5F61 6464 7200 7979 7661 6C00 D		
001DAC2A 5F79 7973 0000 7878 7854 7261 6365 0000 0000 7		
D01DAC44 6567 6973 7465 7200 7870 7274 5F72 6567 6973 7		
DO1DAC5E 6C68 6561 6465 7200 0000 786D 656D 7374 7200 7		
D01DAC78 6B69 7072 6563 6F72 6400 0000 7864 7272 6563 5		
DO1DAC92 7272 6563 5F65 6E64 6F66 7265 636F 7264 0000 7	364 7272 6563 5F63 rrec_endofrecordxdrrec_c	-
39472767 39472767.w		
offset: 1944393 [0x001DAB49]	10 instances of 'Y0' found in 99472767.wbn.0) +B
88IT Signed Byte 50	Address Length	
88IT Unsigned Byte 50	0010E20A 00000002	
ISBIT Signed Short 12851	001BEAC2 00000002	
ISBIT Unsigned Short 12851	001DA6EA 00000002	
32817 Signed Long 842203250 32817 Unsigned Long 842203250	001DAB05 00000002	
Steel Unsigned Long 84/2/03/250	001DABA4 00000002	
6481 Signed Quad 3617235416177049714	001DBC54 00000002	
3280 Float 1.0419273e-008	001DBC60 00000002 001F6756 00000002	
648IT Double 7.048138e-067	0000002	
Data Inspector / Structure Viewer /	Compare A Checksum A Find A Bookmarks A Output	
d Al Complete.		
	Offset: 001DAB49 Value: 12851 2232640 bytes OVR MO	D READ

Figure 4 – Identifying the Power PC Processor

We have loaded the file into the IDA PRO disassembler and debugger program and rebased the image. Now after a little scripting to add the rest of the symbols into our firmware image we can get down to business on understanding the rest of the firmware file structure and any validation routines it might have in place. The function nv_RamValidateChecksumsWriteFlash that makes two calls to ffs_CalcChecksum looks interesting, and here is where we begin to really understand the inner workings of the device.

TIDA - C:\Documents and Settings\dpeck\Desktop\Field Device Research	Rockwell\Firmware\PN-19870.wbn.idb (PN-19870.wbr
File Edit Jump Search View Options Windows Help	
	ffs_CalcCheck 💌 🔗 📄 📼 + 🗙 🔁 🗃 🛄
🗎 🖩 🧄 🗒 🔛 💱 🛦 🎒 🖌 🖩 📄 🏦 🛚	I ™
A En 000 001 001 001 "s" → ★ N × 955 → # · 'x' S H K /·	-/~ 2 :;燕翠 哇翫 聶燕平
IDA View-A 🔄 IDA View-B 🔛 Hex View-A 🔛 Hex View-B 🎦 Exports I	🎇 Imports 📄 Problems 🛛 🔪 Names 🏹 Functions 🕅
😫 IDA View-A	
	nv_RamValidateChecksunsWriteFlash:
	.set var_30, -0x30 .set var_24, -0x24
	.set var_20, -0x20 .set var_10, -0x10
	.set var_C, -0xC .set var_8, -8
	.set var_4, -4 .set arg_4, 4
	stwu %sp, -0x40(%sp)
	mflr %r0 stw %r29, 0x40+var_C(%sp)
	stw %r30, 0x40+var_8(%sp) stw %r31, 0x40+var_4(%sp)
	stw %r0, 0x40+arg_4(%sp) lis %r9, 0x33 # 0x33210C # '3'
	addi %r31, %r9, 0x21BC # 0x3321BC lwz %r3, 0xC(%r31)
	li %r4, 0x14 addi %r3, %r3, 2
	bl ffs_CalcChecksum mr %r29, %r3
	1wz %r4, 0x0(%r31)
	li %r5, 0x16
	bl mencpy 1hz %r0, 0x40+var_30(%sp)
	li \$r30, 0 cmpw \$r29, %r0
	bne 1oc_1421B0
	luz %r0, 0xC(%r31) luz %r3, 0x40+var_24(%sp)
	luz %r4, 0x40+var_20(%sp)
	bl ffs_CalcChecksum
 ۱	1hz %r0, 0x40+var_1C(%sp)
Database for file 'PN-19870.wbn.0' is loaded. Compiling file 'C:\Program Files\IDA\idc\ida.idc' Executing function 'main'	
Executing function 'main' IDAPython version 1.0.0 final (serial 0) initialized Python interpreter version 2.5.2 final (serial 0)	
ROM:001CDC6C: Can't find name (hint: use manual arg) ROM:001CDD4C: Can't find name (hint: use manual arg) ROM:001CDE94: Can't find name (hint: use manual arg)	
	las de Galleri de Hillerel de secolo de la construcción de la destrucción d
^U: idleDown _Disk: 33GB _Click on node title to select/drag it; DblClick on ed	lge to follow it; Wheel to scroll vertically; Ctrl,Alt,Shift for more

Figure 5 – nv_RamValidateChecksumsWriteFlash Function

Reverse engineering the disassembly we find 2 checksums. A 2 byte sum or 2 byte halfwords of the header, and a 2 byte sum of 2 byte halfwords of the remainder of the

file. So we have the header of the file, 2 byte header checksum at 0x0, 4 bytes data offset at 0xc, 4 byte data size at 0x10, and 2 byte data checksum at 0x14. With this knowledge we are able to change the external "wrapper" portion of the firmware as we see fit, this include the web pages stored there, and upload it with the ControlFlash program provided by the vendor so customers can load new firmware.

H Hex Workshop - [PN-19870.BAK]			_ D ×
	THE BSLOF		
与~≪ ≫ 쓸 꼳 쏟 꼳 ▲ ㅣ & ᡟ- +	– 🛎 🖊 % [<] [>] A↑ a.	1 参 昭 昭 % 参 画	
u '			
00000000 EBA1 0000 0000 0000 000	0000 0000 0016 001B	FF00 ED70 4800 004DpHM	-
0000001A 4800 0089 436F 7079 726		3230 3037 2041 6C6C HCopyright (c) 2007 All	
00000034 656E 2D42 7261 646C 657	2043 6F6D 7061 6E79		
0000004E 416C 6C20 5269 6768 747			
00000068 8AA6 6063 0000 3C60 040		3C80 8000 3884 0190 c< x8	
00000082 3CA0 8000 38A5 0100 3CC			
0000009C 03A6 4E80 0020 4800 000	7C84 2278 7C80 0124	7C84 2278 3CAO 55CDN H . "x \$. "x<.U.	
000000B6 38A5 AA33 90A4 0300 3CA	0000 38A5 0002 90A4	0200 7C84 2278 3CA0 83<8 ."x<.	
000000D0 4C00 38A5 0064 90A4 090	7C95 23A6 7C94 22A6	3C80 0000 3884 0400 L.8d .#. .".<8	
000000EA 7C90 8BA6 7C98 8BA6 3C8	0000 3884 0A00 7C90	8BA6 7C98 8BA6 3C80 <8	
00000104 0000 3884 0C00 7C90 8BA	7C98 8BA6 7C00 02E4	4C00 012C 3CA0 00008	
0000011E 38A5 0007 7CBE 23A6 3C8	0000 3884 1002 7080	0124 3C80 FF00 3884 8 .#.<8 \$<8.	
00000138 0000 7C9E 9BA6 7C9E 9AA	5484 001E 3CCO 8000	38C6 0428 3CE0 8000	
00000152 38E7 0528 7CA6 3850 7CA	1670 3CEO 8000 38E7	0100 3D00 8000 3908 8(.8P p<8=9.	
0000016C 0100 7CC7 3050 7CC6 421	3D20 0080 3929 0000	7CA5 4A14 8146 0000 OPI.B.=9)JF	
00000186 9144 017C 9124 0168 38C	0004 3929 0001 7009		
000001A0 8000 3863 0801 9064 010			
000001BA 3863 00C1 9064 0108 3C6			
000001D4 0000 9064 0110 9064 011			
000001EE 9064 0128 9064 012C 906	0130 9064 0134 9064		
00000208 0100 B064 017A 3C60 55C			
00000222 9064 0284 3060 0080 386			
00000230 0000 3806 0001 7006 280			
	FFF8 3C60 8080 3863		
		4082 FFF8 3C60 80808<8	-1
	PN-19870 🕅 PN-19870		
bffset: 0 [0×00000000]		Bookmarks	- 4
8BIT Signed Byte -21	×	Address Length Description Value	
88IT Unsigned Byte 235			
18BIT Signed Short -5215			
ISBIT Unsigned Short 60321			
328/7 Signed Long -341770240			
328/7 Unsigned Long 3953197056 648/7 Signed Quad -1467892003546071040			
⁶⁴⁸¹⁷ Unsigned Quad 16978852070163480576			
328/7 Float -3.8927411e+026			
548IT Double -2.7944472e+210	•		
Data Inspector & Structure Viewer		Compare A Checksum A Find Bookmarks (Output /	
			-
eady		Offset: 00000000 Sel: 0x16 bytes 1834774 bytes OVR MOD	IREAD

Figure 6 – Firmware Header and Checksum

The authors developed proof of concept shellcode that makes use of the "PING" function already in the 1756 ENBT module. This is much like making use of a system call on a Windows operating system. Being careful not to disrupt the current functions of the device we have embedded this code into a portion of the FTP server code. The 1756 ENBT module now pings a specified IP address while continuing to perform its designed purpose for the Logix PAC.

This ping is meant only as a demonstration that an attacker with logical access could load and run arbitrary firmware on the 1756 ENBT module due to lack of any source or data authentication. Since the card has significant unused memory and processing power, an attacker could load and run much more substantial and dangerous code on the module within a reasonable amount of development time. The vendor's ControlFlash program was used to make the upload easier, but we could have developed our own flash upload utility if necessary.

The process of learning and exploiting the load with Koyo/AutomationDirect H4-ECOM100 module was similar. In fact, the process was similar for all of the field devices in our lab that had a firmware upload lacking in source and data authentication.

With firmware images and utilities found on AutomationDirect.com website and the commonly used packet capture utility, Wireshark, we find that the firmware is upgraded using a protocol that Wireshark doesn't know about. We need more information on this upload protocol. One thing that is common in all the messages is they begin with "HAP".

0010 00 a7 00 04 00 07 f1 11 e5 08 c0 a8 0a 83 c0 a8	_	_firmware.pcap - Wiresha					<u>_0×</u>			
Bit I Expression Clear April No. The Surce Desthadon Protocol Brid 1	Ele Ed									
No. The Source Pethodion Pethod Pethod <th></th> <th></th> <th></th> <th>◆ � 중 쇼 E B</th> <th>ାର୍ର୍</th> <th>Q, [] ∰ ⊠ 15 % [2]</th> <th></th>				◆ � 중 쇼 E B	ାର୍ର୍	Q, [] ∰ ⊠ 15 % [2]				
113:06:13.19228 192:168.10.103 125:255.255 100P Source port: 32784 Destination port: 28784 117 13:06:13.193470:192:168.10.105 255.255.255 UOP Source port: 32784 Destination port: 28784 117 13:06:13.193470:192:168.10.105 255.255.255 UOP Source port: 32784 Destination port: 28784 129 13:06:13.19470:192:168.10.105 255.255.255 UOP Source port: 32784 Destination port: 28784 13:06:13.19470:192:168.10.105 255.255.255.255 UOP Source port: 38784 Destination port: 28784 21:31:06:13.1972:41 192:168.10.105 255.255.255.255 UOP Source port: 38784 Destination port: 28784 22:13:06:13.1972:41 192:168.10.105 255.255.255.255 UOP Source port: 38784 Destination port: 28784 23:13:06:13.19878:198.199:188 10:25.255.255.255 UOP Source port: 38784 Destination port: 28784 24:13:06:13.19878:198.199:198.106 255.255.255.255 UOP Source port: 38784 Destination port: 28784 24:13:06:13.19888:10.105 192.168.10.105 255.255.255.255 UOP Source port: 48784 Destination port: 28784 24:13:06:13.29888:10.105	Efter: Expression Qinar Apoly									
16 13:00:13.19472 192:168.10.131 192:108.10.105 UUP Source port: 28744 Destination port: 28744 18 13:00:13.194406 192:168.10.105 255:255:255:255 UUP Source port: 28744 Destination port: 28744 20 13:00:13.19470 192:168.10.105 255:255:255:255 UUP Source port: 28744 Destination port: 28744 21 13:00:13.19470 192:168.10.105 255:255:255:255 UUP Source port: 28744 Destination port: 28744 21 13:00:13.19470 192:168.10.105 255:255:255 UUP Source port: ddt Destination port: 28744 23 13:00:13.19470 192:168.10.105 255:255:255 UUP Source port: ddt Destination port: 28744 24 13:00:13.194875 192:168.10.105 255:255:255 UUP Source port: ddt Destination port: 28744 25 13:00:13.19421.68.10.105 255:255:255 UUP Source port: ddt Destination port: 28744 26 13:00:13.204040 192:168.10.105 255:255:255 UUP Source port: ddt Destination port: 28744 21:31:00:13.204010 192:168.10.105 255:255:255:255 <td></td> <td></td> <td></td> <td></td> <td></td> <td>Info</td> <td></td>						Info				
<pre>17 13:06:13.193408 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 18 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 20 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 21 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 23 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 24 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 24 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 24 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: ddt Destination port: 28784 25 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.194708 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.294708 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.194718 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.294708 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.294719 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.294719 192.168.10.105 255.255.255 U0P Source port: 28784 26 13:06:13.29471 192.168.10.105 255.255.255 U0P Source port: 28784 27 13:06:13.29471 192.168.10.105 11 192.168.10.105 U0P Source port: 28784 27 13:06:13.29480 192.168.10.111 192.168.10.105 U0P Source port: 28784 28 13:06:13.29480 192.168.10.111 192.168.10.105 U0P Source port: 28784 28 13:06:13.49471 192.168.10.103 11 192.168.10.105 U0P Source port: 28784 28 13:06:13.494704 29 13:06:13.49474 29 13:06:13.49474 29 13:06:13.49474 29 13:06:13.49474 29 13:06:13.49474 29 13:06:13.49474 29 1</pre>										
18 13:00:13.194000 192:168.10.105 100 Source port: 28744 Destination port: 28744 20 13:00:13.194708 192.168.10.105 255.255.255 UP Source port: 28744 Destination port: 28744 21 13:00:13.195970 192.168.10.105 255.255.255.255 UP Source port: 28744 Destination port: 28744 21 13:00:13.195970 192.168.10.105 255.255.255.255 UP Source port: 28744 Destination port: 28744 21 13:00:13.195970 192.168.10.105 255.255.255 UP Source port: 28744 Destination port: 28744 21 13:00:13.195970 192.168.10.105 255.255.255 UP Source port: drt Destination port: 28744 23 13:00:13.20246 192.168.10.105 255.255.255 UP Source port: drt Destination port: 28744 24 13:00:13.20246 192.168.10.105 255.255.255 UP Source port: drt Destination port: 28744 26 13:00:13.20246 192.168.10.105 255.255.255 UP Source port: drt Destination port: 28744 28 13:00:13.20246 192.168.10.105 255.255.255										
<pre>19 13:06:13.194708 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 20 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 21 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 22 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 23 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 2874 24 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 2874 24 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 2874 24 13:06:13.195987 192.168.10.105 255.255.255 UUP Source port: 2874 26 13:06:13.20149 192.168.10.105 255.255.255 UUP Source port: 2874 26 13:06:13.20149 192.168.10.105 255.255.255 UUP Source port: 2874 26 13:06:13.20149 192.168.10.105 255.255.255 UUP Source port: 2874 28 13:06:13.49502 Cisco.73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:18:f6:73:59:80 Cost = 0 Port = 0x8001 29 13 13:06:13.49502 Cisco.73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:18:f6:73:59:80 Cost = 0 Port = 0x8001 20 UUP Cisco.79 192.168.10.103 (192.168.10.103) (192.168.10.105) (192.168.10.105) 20 UUP Cisco.79 192.168.10.103 (192.168.10.105) (192.168.10.105) 20 UUP Cisco.79 192.168.10.103 (192.168.10.103) (192.168.10.105) (192.168.10.105)</pre>										
20 13:06:13.199970 192.168.10.131 192.168.10.105 255.255.255 upp Source port: 28784 Destination port: 28744 21 13:06:13.199970 192.168.10.105 255.255.255.255 upp Source port: 28784 Destination port: 28744 23 13:06:13.197254 192.168.10.105 255.255.255.255 Upp Source port: 28784 Destination port: 28744 24 13:06:13.197264 192.168.10.105 155.255.255.255 Upp Source port: 28784 Destination port: 28744 26 13:06:13.200400 192.168.10.105 125.255.255.255 Upp Source port: 28784 Destination port: 28784 26 13:06:13.200400 192.168.10.105 255.255.255 Upp Source port: 28784 Destination port: 28784 27 13:06:13.200401 192.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 29 13:06:13.200401 192.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 21 13:06:13.200401 192.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 Upp.168.10.105 Up										
21 13:06:13.199987 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 21 13:06:13.197268 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 23 13:06:13.197268 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 24 13:06:13.197268 192.168.10.105 255.255.255.255 UUP Source port: 28784 Destination port: 28784 26 13:06:13.20149 192.168.10.105 192.168.10.105 UUP Source port: 28784 Destination port: 28784 27 13:06:13.20149 192.168.10.105 192.168.10.105 UUP Source port: 28784 Destination port: 28784 28 13:06:13.20149 192.168.10.105 192.168.10.105 UUP Source port: 28784 Destination port: 28784 29 13:06:13.202416 192.168.10.105 192.168.10.105 UUP Source port: 28784 Destination port: 28784 30 13:06:13.207259 192.168.10.105 192.168.10.105 UUP Source port: 28784 Destination port: 28784 31 31:06:13.207259 192.168.10.101 192.168.10.105 UUP Source port: 28784 Destination port: 2874 32 13:06:13.207259 192.168.10.101 192.168.10.105 UUP Source port: 28784 Destination port: 2874										
22 13:06:13.197254 192.166.10.131 192.168.10.105 UDP Source port: 28784 Destination port: 28784 23 13:06:13.197264 192.166.10.105 255.255.255.255 UDP Source port: 28784 DEstination port: 28784 24 13:06:13.197264 192.166.10.105 255.255.255.255 UDP Source port: 28784 DEstination port: 28784 25 13:06:13.200461 912.166.10.105 255.255.255.255 UDP Source port: 2878 Destination port: 28784 26 13:06:13.200461 912.166.10.105 255.255.255.255 UDP Source port: 2878 Destination port: 28784 28 13:06:13.200461 912.166.10.105 255.255.255.255 UDP Source port: 28784 Destination port: 28784 29 13:06:13.200461 912.166.10.105 255.255.255.255 UDP Source port: 28784 Destination port: 28784 20 13:06:13.200461 912.166.10.105 UDS Source port: 28784 Destination port: 2474 21 13:06:13.200461 912.166.10.105 UDS Destination port: 28784 Destination port: 28784 21 13:06:13.200461 912.166.10.105 UDS Destination port: 28784 Destination port: 28784 21 13:06:13.200461 012:06.7										
23 33:00:13.197208 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 24 5140:418 193258 4274 555 00.418 1 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 25 13:06:13.20148 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 26 13:06:13.20148 192.168.10.105 255.255.255 UUP Source port: ddt Destination port: 28784 28 13:06:13.20148 192.168.10.113 1 192.168.10.105 UUP Source port: 38784 Destination port: 28784 28 13:06:13.20148 192.168.10.113 1 192.168.10.105 UUP Source port: 38784 Destination port: 28784 28 13:06:13.20216 192.168.10.113 1 192.168.10.105 UUP Source port: 28784 Destination port: 28784 28 13:06:13.49216 (cico73:59:11 Spanning-tree-(for-br STP RST. Root = 32771/00:16:173:59:80 Cost = 0 Port = 0x8001 30 13:06:13.49929 (cico73:59:11 Spanning-tree-(for-br STP RST. Root = 32771/00:16:173:59:80 Cost = 0 Port = 0x8001 31 13:06:15.49929 (cico73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:16:73:59:80 Cost = 0 Port = 0x8001 32 13:06:15.49929 (cico73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:16:73:59:80 Cost = 0 Port = 0x8001 31 19:06:15.49929 (cico73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:16:73:59:80 Cost = 0 Port = 0x8001 32 13:06:15.49929 (cico73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:16:73:59:80 Cost = 0 Port = 0x8001 31 19:06:15.49929 (cico73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:16:73:59:80 Cost = 0 Port = 0x8001 32 13:06:15.49929 (cico35:00), BST: Dell_99:152:2C (00:16:09:99:152:2C) 32 Internet Protocol, Src: 192.168.10.131 (192.168.10.131, DST: 192.168.10.105 (192.168.10.105) 33 USE Datargam Protocol, Src: 192.168.10.131, DST: 192.168.10.105 (192.168.10.105) 34 USE Datargam Protocol, Src: 192.168.10.033 (d0 0.00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0										
24 14/24/03/15 12/24/15/25/04/01 12/24/15/25/04/01 12/24/15/25/04/01 12/24/15/25/04/01 25 13:06:13.198877.1987.168.10.105 255.255.255.255.255 10/P Source port: 28784 Destination port: 28784 26 13:06:13.204871.1997.168.10.105 255.255.255.255.255 10/P Source port: 28784 Destination port: 28784 27 13:06:13.204471.105 255.255.255.255.255 UOP Source port: 28784 Destination port: 28784 29 13:06:13.204471.107.105 255.255.255.255 UOP Source port: 28784 Destination port: 28784 30 13:06:13.204871.101.105 255.255.255.255 UOP Source port: 28784 Destination port: 28784 31 13:06:13.204871.101.11 122.168.10.105 UOP Source port: 28784 Destination port: 28784 31 13:06:13.4989407.156.07.3759:81 Spanning-tree-(for-br SFP RST. Root = 32771/00:1e:f6:73:59:80 Cost = 0 Port = 0x8001 31 13:05:15.499:262 Ciss.10.131 122.168.10.131 Destination										
23 13:06:13.19875 192.168.10.103 255.255.255 UP Source port: ddt Destination port: 28784 26 13:06:13.20140 192.168.10.105 255.255.255 UP Source port: ddt Destination port: 28784 27 13:06:13.20240 192.168.10.105 255.255.255 UP Source port: ddt Destination port: 28784 28 13:06:13.20240 192.168.10.105 255.255.255 UP Source port: ddt Destination port: 28784 29 13:06:13.20240 192.168.10.105 255.255.255 UP Source port: ddt Destination port: 28784 20 13:06:13.20240 192.168.10.113 192.168.10.105 UP Source port: 28784 Destination port: 28784 30 13:06:13.20240 192.168.10.113 192.168.10.105 Source port: 28784 Destination port: 28784 30 13:06:13.20240 192.168.10.113 192.168.10.105 Source port: 28784 Destination port: 28784 30 13:06:13.20240 192.168.10.113 192.168.10.105 Source port: 28784 Destination port: 28784 30 13:06:13.20240 192.168.10.105 Source port: 28784 Destination port: 28784 30 13:06:13.20240 192.168.10.113 192.168.10.105 Source port: 28784 Destination port: 28784 30 13:06:11.459729 Cisco_73:59:81 Source port: 98.1.68.10.131, Dst: 192.168.10.105 (192.168.10.105) 30 19 0.27 00 00 00 00 00 00 00 00 00 00 00 00 00										
26 13:06:13.201430 192.168.10.105 192.155.255 UOP Source port: 28784 Destination port: 28784 28 13:06:13.201430 192.168.10.105 255.255.255.255 UOP Source port: 28784 Destination port: 28784 28 13:06:13.201430 192.168.10.105 255.255.255.255 UOP Source port: 28784 Destination port: 28784 29 13:06:13.201234 192.168.10.105 255.255.255.255 UOP Source port: 28784 Destination port: 20744 29 13:06:13.201234 192.168.10.105 255.255.255.255 UOP Source port: 28784 Destination port: 20744 29 13:06:13.459102 Cisco_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:18:1673:59:80 Cost = 0 Port = 0x8001 21:13:06:13.459102 Cisco_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:18:167:73:59:80 Cost = 0 Port = 0x8001 21:13:06:13.459102 USI:06:06:06:06:06:05:00), DSI: Dell_99:b2:2c UO:16:09:99:b2:2c) USI:06:10:100, USI:199:108 USI:108:108:108 USI:108:108:108:108:108 USI:108:108:108:108:108:108:108:108:108:108						Source port: ddt Destination port: 28784	_			
28 13:06:13.2020.16 192.168.10.131 192.168.10.105 UDP Source port: 28784 Destination port: 28744 30 13:06:13.232732 912.168.10.105 255.255.255.255.255.255.255.255.255.255						Source port: 28784 Destination port: ddt				
29 13:06:13.2327259 192.168.10.105 255.255.255 UOP Source port: ddt Destination port: 28734 30 13:06:13.23847 199.168.10.113 192.168.10.105 UDP Source port: ddt Destination port: ddt 31 13:06:13.45967 2(scc_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:12:16:173:59:80 Cost = 0 Port = 0.8001 22 13:06:13.45967 2(scc_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:12:16:173:59:80 Cost = 0 Port = 0.8001 21 Ertemet 1, src: Hosting1.60:35:00 (O0:e0:65:20:03:60), Dst: Dell_99:162:2c (O0:13:09:99:162:2c) E Thermet 1, src: Hosting1.60:35:00 (O0:e0:65:20:03:5:00), Dst: Dell_99:162:2c (O0:13:09:99:162:2c) E Internet Protocol, src: 192.168.10.131 (192.168.10.131), Dst: 192.168.10.105 (192.168.10.105) E User Datagram Protocol, Src: 192.168.10.131 (192.168.10.131), Dst: 192.168.10.105 (192.168.10.105) E User Datagram Protocol, Src: 192.168.10.131 (192.168.10.131), Dst: 192.168.10.105 (192.168.10.105) E Data (139 bytes) 0000 00 1d 00 99 05 2; C 00 0d 62 60 33 d0 08 00 45 00					UDP					
30 13:06:13.28407 192.168.10.131 192.168.10.130 UP Source port: 28784 Destination port: ddt 31 13:06:13.45926 (sice_7.3:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:16:176:73:59:80 Cost = 0 Port = 0x8001 B Frame 24 (181 bytes on wire, 181 bytes captured) E thermet I, src: Hosting1_60:35:00 (00:e0:62:60:33:00), bst: Dell_99:b2:2c (00:1d:09:99:b2:2c) H Internet Protocol, src: 192.168.10.131 (192.168.10.131), bst: 192.168.10.105) B user batagram Protocol, src Port: 28784 (28784), bst Port: ddt (1052) H Data (139 bytes) 0000 00 1d 09 99 b2 2c 00 e0 62 60 33 00 06 00 45 00 b 5E. 0020 0a 69 70 70 04 1c 00 93 d2 10 45 41 50 0a 00 1a .ipp b4P 0020 00 00 00 00 00 00 00 00 00 00 00 00	2	28 13:06:13.202616	192.168.10.131	192.168.10.105	UDP	Source port: 28784 Destination port: ddt				
<pre>31 13:06:13.459202 Cisco_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:12e:f6:73:59:80 Cost = 0 Port = 0x8001 32 13:06:15.45922 Cisco_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:12e:f6:73:59:80 Cost = 0 Port = 0x8001 # Frame 24 (131 bytes on wire, 131 bytes captured) # Ethernet II. Src: HostEngl_80:55:d0 (Oxee0.45:04:05:04) Dist: Dell_99:b2:2c (Ov:1d:09:99:b2:2c) # User Datagram Frotocol, Src: 192.168.10.131 (192.168.10.131), Dst: Dell_99:b2:2c (Ov:1d:09:99:b2:2c) # Data (139 bytes) D000 00 1d 09 99 b2 2c Op e0 62 60 33 d0 08 00 45 00 b 5E. D0010 00 af 70 00 40 00 f0 11 50 60 00 845 00 b 5E. D0000 00 1d 09 99 b2 2c Op e0 62 60 33 d0 08 00 45 00 b 5E. D0010 00 af 70 00 40 00 f0 10 00 00 00 00 00 00 00 00 D0010 00 af 70 00 00 00 00 00 00 00 00 00 00 00 00</pre>	2	29 13:06:13.237259	192.168.10.105	255.255.255.255	UDP	Source port: ddt Destination port: 28784				
32 13:00:15.499729 (1sc0_73:59:81 Spanning-tree-(for-br STP RST. Root = 32771/00:18:16:73:59:80 Cost = 0 Port = 0x8001 # Frame 24 (181 bytes on wire, 181 bytes captured) # Ethernet IT, Src: Hosting[-60:35:00 (Oce0e162:60:33:00), DST: Dell_99:b2:2c (Oc:1d:09:99:b2:2c) # Internet Protocol, Src: 192.168.10.131 (192.168.10.131), DST: 192.168.10.105 (192.168.10.105) # User Datagram Protocol, Src: 192.168.10.131 (192.168.10.131), DST: 192.168.10.105 (192.168.10.105) # User Datagram Protocol, Src: 202.168.10.131 (192.168.10.131), DST: 192.168.10.105 (192.168.10.105) # Data (139 bytes) Doto 1 d 09 99 b2 2c 00 e0 62 60 35 d0 08 00 45 00 b' SE. DOUD 00 a7 00 00 40 00 ff 11 e5 06 c3 80 88 3C 08 8						Source port: 28784 Destination port: ddt				
B Prame 24 (181 bytes on wire, 181 bytes captured) B Themmet TI, Src: HostEngl_60:35:00 (00:e0:62:60:33:00), Dst: Dell_99:b2:2c (00:1d:09:99:b2:2c) B Internet TI, Src: HostEngl_60:35:00 (00:e0:62:60:33:00), Dst: 192.168.10.105 (192.168.10.105) B User Datagram Protocol, Src Port: 28784 (28784), Dst Port: ddt (1052) B Doto 0 1d 09 99 b2 2c 00 e0 62 60 35 d0 08 00 45 00, b 5E. 0000 00 1f 09 99 b2 2c 00 e0 62 60 35 d0 08 00 45 00, b 5E. 0010 00 a7 00 00 40 00 ff 11 s 5 08 c0 a8 0 48 30 a8, 6										
<pre>B Ethernet II, src: HostEngl.60:35:00 (00:00:02:03:00), Dst Dell_99:b2:2c (00:1d:09:99:b2:2c) B User Datagram Protocol, src: 192.168.10.131 (192.168.10.131) B User Datagram Protocol, src: Port: 28784 (28784), Dst Port: ddt (1052) B User Datagram Protocol, src: Port: 28784 (28784), Dst Port: ddt (1052) B Data (139 bytes) 0000 00 1d 09 99 b2 2c 00 e0 62 60 33 0d 08 00 45 00, b 5E. 0000 00 1d 09 99 b2 2c 00 e0 62 60 33 0d 08 00 45 00, b 5E. 0000 00 1d 09 09 b2 2c 00 e0 62 60 33 0d 08 00 45 00, b 5E. 0000 00 40 00 ff 11 e5 08 c0 a8 0a 83 c0 a8, 6</pre>	3	32 13:06:15.459729	Cisco_73:59:81	Spanning-tree-(for-br	STP	RST. Root = 32771/00:1e:f6:73:59:80 Cost = 0 Port = 0x8001	-			
0010 00 a7 00 00 40 00 ff 11 e5 08 c0 a8 0a 83 c0 a8 e. 0020 0a 69 70 70 40 1 00 93 d2 10 45 41 5 00 a0 1a fpp HAP 0030 1b 82 00 00 00 12 00 00 00 00 02 00 02 00 02 00 01 00 01 e. 0040 00 00 00 00 c2 64 400 cc a3 80 20 0e2 a3 02 00 00 e. 0055 00 00 00 00 c2 64 00 00 00 00 00 00 00 00 00 00 00 e. 0056 00 00 00 00 00 00 00 00 00 00 00 00 00	⊞ Inte ⊞ User	ernet Protocol, Sr r Datagram Protoco	c: 192.168.10.131 (192	.168.10.131), Dst: 192	.168.10.1					
0040 00 00 00 00 00 00 00 00 00 00 00 00	0010	00 a7 00 00 40 00 0a 69 70 70 04 1c	ff 11 e5 08 c0 a8 0a 00 93 d2 10 48 41 50	83 c0 a8	HAP					
0055 00 00 00 00 00 00 00 00 00 00 00 00										
0066 0 00 00 00 00 00 00 00 00 00 00 00										
0070 00 00 00 00 00 00 00 00 00 00 00 00										
0088 00 00 00 00 00 00 00 00 00 00 00 00										
2000 00 00 00 00 00 00 00 00 00 00 00 00										
	00a0	00 00 00 00 00 00								
The William water and Particular Manufacture Conductor 1970 Workshol 0	0060	00 00 00 00 00 00								
	talau No. Je	Page marks and Collinear dead	Dadate: 1020 Diala	nd: 1970 Muladi 0		Paulta Para				

Figure 7 – Identifying "HAP" in Messages

We could not find information on this protocol on the Koyo or Automation Direct websites. However, an Internet search revealed a document that described the protocol and ECOM interface in some detail [3], see Figure 8.

	ew Document Comments Fo	rms Tools	Advanced	Window	Help									
Create	PDF 🔹 🕌 Combine Files 🔹	Expor	t 🕶 🔊 S	Start Meetin	g • 🔒	Secure 🔻	🖉 Sign 🔻	🔗 Re	view & Corr	ment •				
	🖹 🖃 🛛 🔶 📔	2 / 11	IR 🤇	9 🔍		101%	• 日		Find	•				
	CHAPTER 1. ECC	M ETH	ERNET	INTER	FACES	PECIFIC	CATION						:	2
			"HAP"	,	ap	p. id	CI	RC	len	gth	DATA			
		0	1	2	3	4	5	6	7	8	9+			
	"HAP" in ASCII ((packet sequence nu requests will have	umber. T their sec	ypically Juence n	this sta umber e	rts at on equal to	e and in the sequ	creases fo ience nu	or each mber of	new pac the requ	ket sen uest. Tl	t Packets th ne CRC is o	at are responsed	ponses to using a	n 1
	packet sequence nu	umber. T their sec nplemen of the d ta section or most r ed. Pado ections th endian address A	ypically quence n ts a mov ata section begins requests, ded bytes hat all m (MSB fir AA:BB:C	this star umber e ving excl on is dro with a 1 additio s should ulti-byte st). Spec C:DD:E	rts at on equal to lusive-or opped (ig l6-bit me nal bytes l be zero e fields a cifically, E:FF, the	e and ine the sequ r and a ti gnored). essage (o s may be and omi are in litt for the II bytes ar	creases for tence nut ranslation The final r comma e specifie titted byte le-endiar ? address e stored :	or each : mber of n table. l field of und) nur d (for 10 es are as n notations in notations in the or in the or	new pack the requ A packed f the heat mber. The 6- or 32- sumed z on (LSB) the byte rder: AA	ket seni uest. Tl et whose der is tl ie list of bit pade ero. first). C is are ste ., BB, CO	t Packets the cRC is de cRC does the length of messages, ding) and s ertain field ored in the C, DD, EE,	at are responsed computed is not mate the data shown in ometimes s require order: 1, 2	ponses to using an h agains section o <i>Table 1.1</i> required "networl , 3, 4. Fo	n t f
	packet sequence nu requests will have algorithm which in the computed CRC the packet. The dai is detailed later. Fc bytes may be omitt Note in these se order" which is big the ethernet MAC is	umber. T their sec nplemen of the d ta section or most r ed. Padd ections th -endian address <i>I</i> resses loca	ypically puence n ts a mov ata section begins requests, led bytes nat all m (MSB fir AA:BB:C ally, that	this star umber e ving excl on is dro with a 1 additio s should ulti-byte st). Spec C:DD:E	rts at on equal to lusive-or opped (ig l6-bit me nal bytes l be zero e fields a cifically, E:FF, the	e and ine the sequ r and a ti gnored). essage (o s may be and omi are in litt for the II bytes ar	creases for tence nut ranslation The final r comma e specifie titted byte le-endiar ? address e stored :	or each : mber of n table. l field of und) nur d (for 10 es are as n notations in notations in the or in the or	new pack the requ A packed f the heat mber. The 6- or 32- sumed z on (LSB) the byte rder: AA	ket seni uest. Tl et whose der is tl ie list of bit pade ero. first). C is are ste ., BB, CO	t Packets the cRC is de cRC does the length of messages, ding) and s ertain field ored in the C, DD, EE,	at are responsed computed is not mate the data shown in ometimes s require order: 1, 2	ponses to using an h agains section o <i>Table 1.1</i> required "networl , 3, 4. Fo	n t f

Figure 8 – ECOM Protocol Packet Header

Examining the firmware images we get some clues about the internals of the module. "Nios OS" is a good bit of information that its run on the NIOS embedded FPGA processor from Altera. Looking further into the firmware upload procedure we find that the upgrade program, netedit, seems to work over UDP broadcast protocol. However with further investigation we find that this is a weakness of the program and not a weakness of the protocol itself.

We find the same lack of source and data authentication on firmware uploads in the Koyo system as found in the Rockwell Automation PAC. In fact the Koyo PLC firmware upload appears to even lack a checksum. As a proof of concept for malicious firmware upload, we uploaded modified web pages that are available from the ECOM module.

While it is bad enough that these systems can be so easily compromised, an even greater concern is as long as the compromised field device is still functioning as expected the asset owner could be completely oblivious to the fact that many or even all of the field devices in the control system have been compromised.

Extrapolating out from this and working with other devices we have found similar results. Over the network firmware uploads are, in general, not authenticated or otherwise protected in any meaningful way. Typically a checksum is used to authenticate

the integrity from a non-malicious, load integrity standpoint. This checksum offers little protection from a motivated and moderatedly skilled attacker.

Many field devices use different and more obscure processors than what an average attacker would see in the PC-dominated corporate world. While the obscurity of a processor can make the potential attackers work harder to investigate, develop and implement a malicious firmware upload to a field device, it is not a safeguard. Much work has been done on similar platforms by the gray and black hat community, such as home switches and routers, and that knowledge will transfer over easily for future field device hackers.

Even this security by obscurity is becoming less of a deterrent. Many field devices are moving to Intel x86 chips as well as more common operating systems such as embedded XP or embedded Linux. While these are not necessarily easier to develop firmware uploads for, there are many more attackers and tools available to help an attacker exploit these systems. It will take less specialized knowledge and less expensive tools to attack a field device on an x86 platform as compared to a field device with an ARM.

2.3 Attacking the PLC

As mentioned in the Introduction section, logical access to most field devices allows an attacker to read and write to the device. So what benefit is there in uploading new firmware over directly monitoring the sensors and controlling the actuators and other instruments in real time? The answer to this question is limited only by imagination. Here are a few scenarios to illustrate the benefit to an attacker.

2.3.1 Delayed Attack

An attacker may want to stage an attack in advance and have it ready to launch at a time that is particularly damaging. Imagine an attack on a region's power system that caused an outage at 11:58 on New Years Eve, on election day, or at the start of a big event like the Olympics. The malicious firmware upload could allow normal operations until a certain time and date and then launch a scheduled attack.

A delayed attack scenario could be coordinated between multiple field devices that had been exploited with a malicious firmware upload that set up a cron job to run at a specific time. These could be field devices across a single control system or even field devices in multiple control systems.

Another benefit of the delayed attack is it would be substantially harder for an asset owner to determine what caused the outage or incident. A review of recent traffic to and from the field device, if this were even available, would not show the firmware upload. Depending on the nature of the attack, there is a good chance that the asset owner would not even realize a cyber attack took place. It could be made to look like a hardware failure, communications error or some other non-malicious cause.

Finally, a delayed attack can address the situation of an attacker losing access to the field device. Imagine a malicious firmware upload that required a message from the attacker on a monthly basis or an attack would be launched. This attacker may want to have the capability to bring down part of the critical infrastructure or a particular organization he

holds a grudge against, but chooses to reserve that capability for an unspecified future need. If the asset owner has improved perimeter security and the attacker no longer has access to field devices he owned. His capability to schedule an attack is lost, but the attack would launch within a month.

2.3.2 Brick the Ethernet Card

The malicious firmware could be programmed for an Ethernet card failure that required a return to factory by removing the ability to reload new firmware and removing the proper firmware capabilities - - turning the card into essential dead weight or a brick.

A single Ethernet card failure is unlikely to cause much of a problem for an organization mindful of redundancy and recovery requirements in a control system. Spares would be available. However it will take time to replace the card, especially in a geographically dispersed SCADA system. If this attack was made on Ethernet cards in a large number of controllers this could exhaust the spares and cause availability problems to travel and replace numerous simultaneous Ethernet card failures.

2.4 "Random" Attack or Failures

An attacker's goal may not be to cause an outage that affects a large number of people. The attacker may not want to shut down power to a region or affect the ability of a large city to get gasoline or water. This level of impact may be more than the attacker desires and the consequences of getting caught performing such an attack is likely to prevent many potential attackers as discussed in the Langner and Singer paper [4].

For an attacker, perhaps a disgruntled insider, who just wants to make difficulties for the operators and engineers a "random" attack or failure mode may be more interesting. The attacker could load malicious firmware into multiple field devices and have a pseudo random number generator select the attack and the attack time. Imagine the case where the malicious software could temporarily make the device unavailable, brick the card, send false data back to the control center, make process changes or simply reboot the system multiple times. The random number could select the attack from the list, a second number could select the attack start time and date, and a third random number could select the attack duration.

Troubleshooting these problems could be made very difficult if the field device returned the expected information, such as version number and checksum, during diagnostic tests. The malicious software could also prevent the loading of new firmware onto the Ethernet card without a special code or authentication mechanism. The malicious firmware could be programmed to issue a response that indicated the asset owner had successfully loaded new firmware, perhaps even modifying the firmware version number returned when requested, while in fact the malicious firmware remained intact.

2.5 A Field Device Worm

Each field device has its own set of software and methods to program, configure and manage the system. Once an asset owner has purchased and learned to configure and program a field device, they are likely to purchase additional field devices of the same

model. While it is not unusual for an asset owner to have a variety of field devices, it is typical that they will have tens or hundreds of each vendor model.

Since the unauthenticated firmware upload allows an attacker to load malicious code on the field device, the attacker may be interested in identifying all field devices of the same model and compromise all of them. This is achievable with a field device worm designed to take advantage of the lack of source and data authentication of a field device firmware upload.

As an example, imagine an attacker has identified an Allen Bradley field device on the network. Since these devices typically use the EtherNet/IP protocol on TCP/44818 they are easy identify on a network with a port scan of that port. A worm can be written that would:

- 1. Port scan to identify other systems with EtherNet/IP ports that are open
- 2. Attempt to load the attacker's firmware on all identified EtherNet/IP systems
- 3. Report back to the master after the system is compromised
- 4. Standby for future instructions or take preprogrammed actions

An attacker would be in a similar situation to attackers that have control of an army of bots except they would be on a control system network.

2.6 Attack Other Cards on the Field Device

While the Ethernet card is the most accessible to an attacker or malware, it is not the most important card in a field device. In fact many control systems today use the field device Ethernet card for maintenance and still rely on serial communication cards for SCADA system monitoring and control. This is likely to be less common over time as more asset owners move to an IP WAN for their SCADA systems.

The most important card in field devices is typically the CPU card. If this card is not functioning properly, the entire field device will not function. Some CPU cards also support a remote firmware update through the Ethernet card. The CPU card may have a different computing architecture, different load utility and other differences from the Ethernet card, but it also may be identical.

Perhaps the simplest attack on the CPU card is a denial of service attack that could brick this CPU card. Imagine a case where the Ethernet card is compromised with rogue firmware that periodically, or perhaps pseudo randomly, uploaded to firmware to the CPU card that bricked the CPU card. The asset owner would replace the CPU card, but the underlying problem would remain in the Ethernet card.

2.7 Attack Other Systems

The attacks discussed in Section 2 to this point have focused on attacking the field devices to affect the monitoring and control of the process. However, one or more compromised systems on a control center LAN or WAN provide an attack platform to go after other systems on the SCADA or DCS, such as workstations or servers.

Field devices are usually not considered as potential attack platforms. They are not patched or upgraded frequently; typical asset owners do not worry about security patches, anti-virus and security configuration issues with field devices nearly as much as they do for a Windows HMI; they are deployed and often not touched for years. In many ways, these field devices are similar to printers on a corporate network. They are computing platforms spread throughout the network that are treated like appliances rather than computing devices.

One of the common findings in control system security assessments is missing security patches. It is not unusual to find Windows systems missing patches for many years. An asset owner who is diligently patching the Windows OS every month may be missing patches in the Oracle database, Java Runtime Environment, Apache web server, or other non-Windows software. An attacker could upload malicious firmware on a field device to identify and exploit missing security patches on workstations and servers on the SCADA and DCS.

An example of this malicious software could be in the form of a worm that would seek out other similar devices to exploit and load itself onto, but also have a set of attacks for HMI's, real time servers, and historians. This behavior could be uncontrolled as many early worms were, and simply exist for destructiveness, or it could be controlled by an authority or "botmaster" directing the activities of the compromised devices and systems.

2.8 Solutions

The solutions to the malicious firmware upload over the network are straightforward. The low-tech solution is to not allow over the network firmware upload. Many controllers have physical keys that can restrict the ability to load new firmware over the network or locally. This is an effective control, but it can have a significant impact on the ability to manage a system of field devices, especially in a geographically dispersed SCADA system.

The high-tech solution is to add source and data authentication to the firmware upload process. The most straightforward way to do this is with a public key certificate infrastructure and a digital signature on the firmware upload. The field device could verify the certificate and the digital signature to confirm the firmware was from an authorized entity and had not been changed in transit. This would require public key implementations on the Ethernet cards, but efficient and secure algorithms are available using elliptic curve technology [5]. Since firmware updates are rare events, the processing time required to verify the certificate and digital signature is not an important factor.

3 Management Application Vulnerabilities

Field device vendors have been trying to find ways to make field device management easier and provide more status information to network management and monitoring systems. It is common to find field devices with web servers, FTP servers, the SNMP service, and the Telnet service. Each of these servers or services are well known and commonly attacked by the hacking community.

Sometimes the operating system on the Ethernet card provides these servers and services; in other cases the field device vendor develops their own server and service; and in many cases the field device vendor buys a low cost or obtains a free management server or service. The commonality in all these cases is these management servers and services are rarely designed to be full featured, maintained and secured. They often fall into the category of "quick and dirty".

An alert asset owner will know they should patch a Microsoft IIS or Apache web server by monitoring one or more cyber security bug fix lists. But that same asset owner is unlikely to even monitor the vendor of the web server on a field device for vulnerabilities and security patches.

In a review of the field devices installed in Digital Bond's lab, we found that many of the common vulnerabilities in similar IT applications in the applications integrated into the field device. This section provides examples of common vulnerabilities in these management services found in a small sample of field devices.

3.1 Web Server

Several different forms of cross site scripting (XSS) were found on the sample field devices. These vulnerabilities can be leveraged to exploit vulnerabilities in client side software, such as Internet Explorer. We found both persistent and non persistent XSS, some that would require social engineering to encourage a user to follow a link, and others that could wait for a user to visit the management interface before the attack was launched, weeks or even months later.

Flaws were also found which enabled unauthenticated users to view the ASP source code of a page and access configuration files without a password by appending a trailing "/" to the URL.

3.1.1 Example

The Koyo H4-ECOM100 module provides a good demonstration of these types of vulnerabilities. The management interface requires no authentication to access the device and the user editable fields are not properly sanitized before being served back to the user.

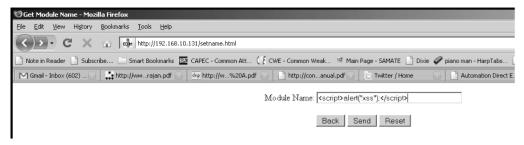


Figure 9 – Part 1 of XSS Example on Koyo H4-ECOM100 Module

10 ECOM100 Main - Mozilla Firefox					
Ele Edt Vew Higtory Bookmarks Icole	Rep				
C > C X & 1 m	(/192.168.10.131/main.html				☆・ C. Google
Note in Reader 📄 Subscribe 📄 Smart B	ookmarks 📴 CAPEC - Common Att 🤇	🕴 CWE - Common Weak 🧉 Ma	in Page - SAMATE 📄 Di	ie 🗳 piano man - HarpTabs	
M Gnai - Inbox (602) 💿 📑 http://www.re	sjan.pdf 🔄 🎰 http://w%20A.pdf	🗟 📄 http://conanual.pdf	C Twitter / Hone	Automation Dire	d E ECOM100 Main
	H4-EC	OM100 by <u>A</u>	utomatio	n Direct.co	<u>om</u>
			Mod	foo ID: 0 (0x0)	
			Module	Name	
		The page of http://150.11	5λ11.133 seys α	X	

Figure 10 – Part 2 of XSS Example on Koyo H4-ECOM100 Module

3.2 FTP Server

During the disassembly of the Rockwell Automation device code a FTP server was discovered, however the device does not have this FTP server functionality enabled. By manipulating the firmware and simply replacing a 0 with a 1 in an XML configuration file that was clearly visible in the firmware we were able to activate and interact with the FTP service. So we have not only expanded the functionality of the device, but also expanded the attack surface. This functionality was provided by the underlying operating system and the vendor may not have been aware that the FTP server code was even on the Ethernet card.

Initially we are simply presented with a login when connecting to the FTP service, but with the Administrator username and blank password, discovered through the assessment of web server, we are able to login to the device. Not only did this open up many other FTP commands to attack, commands in which vulnerabilities were found, but also now provides a location to store information for later use in exploitation or to store damaging or illicit data.

3.3 SNMP

The Rockwell Automation 1756 ENBT module also runs a SNMP service that allows large amounts of information to unauthenticated sources that could be leveraged in further attacks against the system, such as services and types of device. The lack of authentication is just further evidence that these field devices were designed with the

expectation they would be in a secure environment and only accessible by authorized users.

4 Operating System Vulnerabilities

While beyond the scope of this paper, the authors feel it is important to recognize that the underlying field device operating system is a large attack surface that has received little scrutiny by security researchers or the hacking community to date. Some real time and embedded operating systems are used in numerous field devices that monitor and control critical infrastructures. These are likely to garner more attention by directed attackers in the future and are worthy of analysis by security researchers.

5 Conclusion

The Ethernet cards on field devices present an attractive target for attackers because of their distribution throughout a modern control system, lack of security features and lack of security attention paid to these cards. In this paper, the authors have demonstrated how a lack of authentication in the firmware upload allows an attacker to load malicious code into two field device Ethernet cards. The impact of this malicious code on the process, field device and other components on the control system is limited only by the creativity and imagination of an attacker.

Control system asset owners should be determining if firmware can be uploaded into their field device Ethernet cards. And if the answer is yes, is this source and integrity of this firmware upload authenticated prior to loading on the card? If firmware authentication is not required, the vendor should be contacted and strongly encouraged to address this vulnerability. Asset owners should also consider adding authentication of firmware uploads to any future buying criteria.

Finally, control system asset owners and vendors should not ignore the security of the management services commonly found on Ethernet cards such as HTTP, FTP, Telnet and SNMP.

About the Authors – Daniel Peck is an Offensive Security Researcher at Digital Bond, and a key member of the U.S. Government funded Portaledge and Quickdraw research projects. Prior to joining Digital Bond, Mr. Peck was a security researcher at SecureWorks and a frequent presenter at leading security events such as BlackHat, Defcon and the Pentagon Security Forum.

Dale Peterson is a founder of Digital Bond, Inc. and has led their Control System Security Consulting and Research Practice since 2000. He began his career as an award winning Cryptanalyst at the National Security Agency and has more than twenty years experience in information security.

References

- Digital Bond SCADApedia, http://www.digitalbond.com/wiki/index.php/SCADA_IDS_Signatures.
- [2] E. Byres, D. Hoffman and N. Kube, "On Shaky Ground A Study of Security Vulnerabilities in Control Protocols", American Nuclear Society, November 2006.
- [3] ECOM Ethernet Interface Specification, Host Engineering.
- [4] R. Lagner and B. Singer, "SCADA Threat Modeling Using Attack Scenarios", 2008 S4 Proceedings, Digital Bond Press, 2008.
- [5] R. Lambert, "ECC and SCADA Key Management", 2007 S4 Proceedings, Digital Bond Press, 2007.