# Snort Signatures for AB/ML Metasploit Major Fault Attack

**AUTHOR: DERON GRZETICH - DEPAUL UNIVERSITY** 

2013

# Snort Signatures for AB/ML Metasploit Major Fault Attack

# Objective

The objective of this work was to write alert signatures for Snort to detect the Metasploit Fault attack on the Allen-Bradley/Rockwell Automation MicroLogix 1400 series controllers. The first objective was to write a Snort IDS rule that would detect the malicious traffic generated by the exploit. The second objective was to write a rule that identifies only "approved traffic" and alerts on all other traffic by tightening the Snort rule to only data that puts the controller into a fault state.

# Approach

The approach to this work was to examine both the exploit in action through packet captures and analysis, as well as through a review of the Metasploit .rb file which dictates how the packets will be created and sent to the MicroLogix controller. After understanding how the attack functioned a Snort rule was written to detect traffic that matched the exploit traffic from a generic level. As the generic rule may also trigger false positives on legitimate traffic sent to the controller, in addition to the attack, the rule was refined. After refining the rule, the bytes in the CIP Generic Class section of the payload which resulted in a successful fault attack were determined and used to further refine the rule using offset and depth information.

# **Environment Setup**

For the purposes of this report, the environment was setup as follows:



# **Understanding the Exploit**

To better understand the exploit and its operation, the function of the exploit through examination of its usage in metsaploit was conducted. In addition, the ports and services that are "listening" or used by the controller were examined through port scanning, the effect of the exploit on the S2:5/3 bit was determined, and finally the attack was dissected through both packet capture and analysis and examination of the .rb file to match what was observed on the wire.

# Metasploit Examination

The exploit was loaded into Metasploit (located at /auxiliary/cybati/micrologix\_fault). The options for the attack were examined by running a "show options". It appears that the only option that needs to be set is the RHOST, as changing the port or attack type (as examined in the .rb file) modifies the attack in such a way that the exploit will not function as designed. The screenshot below shows the use and options of the attack:

~ ~ ×	root@bt: ~			
File Edit V	/iew Terminal Help			
=[ +=[ +=[ =[	metasploit v4.5.0 974 exploits - 51 261 payloads - 28 svn r15999 update	-dev [core 3 auxiliar encoders d 195 days	:4.5 api:1.0] y - 159 post - 8 nops age (2012.10.22)	
Warning: T W F	his copy of the M e recommend that or information on https://commun	etasploit you update updating ity.rapid7	Framework was last u the framework at le your copy of Metaspl .com/docs/DOC-1306	pdated 195 days ago. ast every other day. oit, please see:
<u>msf</u> > <u>msf</u> > use <u>msf</u> auxil	auxiliary/cybati/ iary(micrologix_f	micrologix ault) > sh	fault ow options	
Module opt	ions (auxiliary/c	ybati/micr	ologix_fault):	
Name	Current Setting	Required	Description	5
ATTACK RHOST RPORT	FAULT 127.0.0.1 44818	yes yes yes	The attack to use. The target address The target port	Valid values: FAULT
<u>msf</u> auxil	iary(micrologix_f	ault) >		

Figure 1: Metasploit MicroLogix attack options

The following screenshot shows successful execution of the attack against the controller (note the connection and session IDs referred to in this document were not captured in this screenshot, this is to illustrate the attack in Metasploit only):



Figure 2: Metasploit MicroLogix attack execution

#### **Port Scanning**

To examine what other ports may be a used in this attack on the MicroLogix controllers a full nmap scan with options (-sT -n -v -p 1-65535) was run against the controller at 10.0.0.135. The results showed two open ports (80/TCP and 44818/TCP) and one closed port (2222/TCP). Port 80/TCP is used for the web interface to the controller, while 2222/TCP is shown as ENIP and 44818/TCP is an unknown service. It is likely that the 2222/TCP closed state is related to the fact that ENIP uses 2222/UDP for implicit messaging and 44818/TCP is used for explicit messaging.

#### Examining the S2:5/3 Bit

To see the effect of the exploit on the controller, and to prove that the S2 file's 5/3 bit is "set" as part of the exploit, a before and after view of the bit status (using the controller's web interface) was used. In the first screenshot below we see that the S2:5/3 bit is off and the controller is running normally without a fault indication.

🖉 1766-L32BWAA B/11.00 - Windows Internet Exp	plorer		* *	-													_ 0	×
			_								-	- 10	× 🖸 E	Bing				<del>ب</del> 0
🙀 Favorites 🛛 🚔 💽 Suggested Sites 👻 🖉 V	Veb Slice Ga	llery 👻																
🏉 1766-L32BWAA B/11.00													- 🖾 -	🖃 🖶 ·	• Page •	<u>S</u> afety ▼	T <u>o</u> ols ▼ (	0-
Allen-Bradley 1766-L32	2BWA	A B/	11.00	)													Rockw	ell
Expand Minimize	Data View	s New	Data View													Au	oman	on
DHome	Doto Her		boto rich	1														16
Data Views	5	2 [State	us] Disp	lay As	Binary	•									Update	Ba	ick	-
New Data View	Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1
User Provided Pages	S2:0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
User #2	52:1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	
User #3	52:2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
User #5	52:3	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
User #6	52:4	0	0	1	o	1	1	1	0	1	1	0	o	1	1	1	0	
User #8	\$2:5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Diagnostics	S2:6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Diagnostic Overview	S2:7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Network Status	52:8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Administrative Settings	S2:9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Server Settings	52:10	0	0	0	0	0	0	0	0	0	0	0	o	0	0	0	0	
C) User Management	S2:11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	52:12	0	o	0	0	0	o	0	0	0	0	0	0	0	0	0	0	
<	52:13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
											😜 I	nternet   Pro	otected Mo	de: Off		- 	<b>a</b> 100%	•

Figure 3: Status of the S2:5/3 bit prior to exploit being run

Next, the exploit was run against the controller which induced the fault condition. The interface was again examined for the presence of the S2:5/3 bit being set. In the screenshot below we can see that the bit is now set and the controller must be manually reset to "reset" the S2:5/3 bit.

A 1766-L32BWAA B/11.00 - Windows Internet Ex	plorer																_ 0	x
			* *	1.0							•		🗙 📴 Ві	ing			,	o •
🙀 Favorites 🛛 🍰 🔽 Suggested Sites 👻 🔊 🛛	Web Slice Gal	lery 🕶																
													• 🔊 •	-		<u>S</u> afety ▼	T <u>o</u> ols 🕶 👔	0-
Allen-Bradley 1766-L32	2BWA	A B/1	1.00	)												Aut	tockwe omatie	ell on
Expand Minimize	Data View	s New D	ata View															
Home Data Views Data Views Data Views	s	2 [Status	a] Disp	olay As	Binary	•									Update	) 🛛 Ва	ck	ш
New Data View	Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
User =1	52:0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	μ
User #2	52:1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	
User #4	52:2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
User #5	52:3	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
User #7	52:4	0	0	1	0	1	1	1	0	1	1	0	0	1	1	1	0	
User #8	\$2:5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
Diagnostics	S2:6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Network Settings	S2:7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Network Status	52:8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Administrative Settings	52:9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
User Management	52:10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	S2:11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	S2:12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
< F	52:13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-
											😜 Ir	nternet   Pro	otected Moo	le: Off		- 10	<b>a</b> 100%	•

Figure 4: Status of the S2:5/3 bit post exploitation

### Examining the Packets

#### Wireshark

To dissect how the attack operates Wireshark was used to capture packets on the network and the results filtered to examine packets sent between Metasploit and the controller as part of the attack. The packets, as show in the screenshot below, have the following characteristics:

The attack requires a total of 12 packets, in order the packets are:

- Packets 1-3: The TCP connection setup (SYN-SYN/ACK-ACK)
- o Packets 4-6: Register an ENIP session to capture the ENIP Session ID
- Packets 7-8: Generate and capture the ENIP Connection ID
- Packet 9: Forge packet and send with CIP data which induces the fault
- Packets 10-12: Close the TCP connection (ACK-FIN/ACK/ACK-RST)

🖊 AB	ML Metasploit At	tack - Packet Cap 1.p	cap - Wireshark		
<u>F</u> ile	<u>E</u> dit <u>V</u> iew <u>G</u> o	<u>Capture</u> <u>A</u> nalyze	<u>Statistics</u> Telephony <u>T</u> ools	<u>H</u> elp	
		🖻 🖬 🗶 🤗	≗   °, ⇔ ⇒ ⇒ 7		Q, Q, Q, [7]   ₩ 12 🥵 %   13
Filter:				<ul> <li>Expression</li> </ul>	. Clear Apply
No.	Time	Source	Destination	Protocol	Info
	1 0.000000	10.0.0.130	10.0.0.135	ТСР	53436 > EtherNet/IP-2 [SYN] Seq=1340407457 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSV=6995
	2 0.005585	10.0.0.135	10.0.0.130	ТСР	EtherNet/IP-2 > 53436 [SYN, ACK] Seq=813180826 Ack=1340407458 win=2000 Len=0 MSS=1478
	3 0.005650	10.0.0.130	10.0.0.135	TCP	53436 > EtherNet/IP-2 [ACK] Seq=1340407458 Ack=813180827 Win=14600 Len=0
	4 0.008346	10.0.0.130	10.0.135	ENIP	Register Session (Req), Session: 0x00000000
	5 0.013086	10.0.0.135	10.0.130	ENIP	Register Session (Rsp), Session: 0x73E5CCAA
	6 0.013099	10.0.0.130	10.0.0.135	TCP	53436 > EtherNet/IP-2 [ACK] Seq=1340407486 Ack=813180855 Win=14600 Len=0
	7 0.014230	10.0.0.130	10.0.0.135	CIP CM	Forward Open
	8 0.023108	10.0.0.135	10.0.130	CIP CM	Success
	9 0.023767	10.0.0.130	10.0.0.135	CIP	Unknown Service (0x4b)
	10 0.024027	10.0.0.130	10.0.0.135	ТСР	53436 > EtherNet/IP-2 [FIN, ACK] Seq=1340407645 Ack=813180925 Win=14600 Len=0
	11 0.028079	10.0.0.135	10.0.0.130	TCP	EtherNet/IP-2 > 53436 [ACK] Seq=813180925 Ack=1340407646 win=2000 Len=0
	12 0.030347	10.0.0.135	10.0.0.130	ТСР	EtherNet/IP-2 > 53436 [RST, ACK] Seq=813180925 Ack=1340407646 Win=2000 Len=0

Figure 5: Wireshark capture of the exploit traffic between Metasploit and the controller

It appears that the forged packet (Packet 9) requires both a session and connection ID to succeed, although these connections do not require authentication prior to being accepted and the attacker is able to send the forged packet to the controller with minimal information being required on their part.

#### Packet and .rb Code Analysis

To further dissect the packets the options and data included with each packet were examined, including ENIP and CIP options, to evaluate how the attack operates. The tables below show the opening of the TCP connection, Session registration and capture of the Session ID, utilizing the Session ID to capture the Connection ID, the attack, and the TCP connection close. In addition, a review of the associated .rb file sections is included where necessary.

The TCP Connection open utilizes a standard open procedure:

	IP Header		TCP Header		
Packet number	Src IP	Dst IP	TCP Flags	Src port	Dst Port
1	Metasploit	Micrologix 1400	S	53436	44818
2	Micrologix 1400	Metasploit	SA	44818	53436
3	Metasploit	Micrologix 1400	A	53436	44818

#### **1. TCP Connection Open**

Table 1: The TCP connection open from the attack

(note IP's have been removed and replaced with system monikers)

The TCP open is simply part of the *sock.put(packet)* requirement that a TCP session be established prior to sending the forged packets. The source port is chosen randomly from ephemeral ports. Although, after several runs of the attack and analysis of the chosen source port it appears that ports above 50,000 are used. The destination port is set by the exploit through the RPORT setting and the target is set by RHOST.

In terms of items to key in on to create a Snort signature we have:

- The source port will be random
- The destination port is that of the ENIP/CIP protocol which is 44818/TCP

Next, the exploit registers an ENIP session with the controller on 44818/TCP in order to generate a Session ID. This Session ID will be captured by the module and used in the next series of packets to capture the Connection ID which is required by the final attack packet. The truncated table below shows some of the options as set by the exploit:

	IP Header		TCP Heade	er		ENIP	ENIP Header	
Packet	Src IP	Dst IP	TCP Flags	Src port	Dst Port	Session	Command	Session
				P				
						0x00000000 Register	0x0065 Register	0×0000000
4	Metasploit	Micrologix 1400	AP	53436	44818	Session	Session	Success
5	Micrologix 1400	Metasploit	AP	44818	53436	0x73E5CCAA Register Session	0x0065 Register Session	0x00000000 Success
6	Metasploit	Micrologix 1400	A	53436	44818	NA	NA	NA

# 2. Register and Capture Session ID

Table 2: Session ID request and capture packets

In terms of actual code in the exploit module used to generate the packet above, we examine the following section of code called *reqsession*:

def reqsession

packet = ""
packet += "\x65\x00" # ENCAP\_CMD\_REGISTERSESSION (2 bytes)
packet += "\x04\x00" # encap\_length (2 bytes)
packet += "\x00\x00\x00\x00" # session identifier (4 bytes)
packet += "\x00\x00\x00\x00\x00" # status code (4 bytes)
packet += "\x00\x00\x00\x00\x00\x00\x00\x00" # context information (8 bytes)
packet += "\x00\x00\x00\x00\x00" # options flags (4 bytes)
packet += "\x01\x00" # proto (2 bytes)
packet += "\x00\x00" # flags (2 bytes)
begin
sock.put(packet)
response = sock.get\_once(-1,8)
session\_id = response[4..8].unpack("N")[0] # parse allocated session id
print\_status("Got session id: 0x"+session\_id.to\_s(16))

TRUNCATED...

The code above generates the portion of the packet used to generate a Session ID through the 0x0065 (Register Session) value in the command field of the ENIP header. The variable "packet" is built up with other options such as a session identified (0x0), status code (0x0 Success), and other options required by the ENIP header. Again, full packet analysis is included as an attachment. The module then sends the

packet using *sock.put(packet)* to the controller and evaluates the response. The *response* = *sock.get\_once(-1,8)* receives the packet back from the controller, and the "session\_id" variable is populated with the parsed response packet which contains the Session ID. If successful it prints the Session ID to the console, and the truncated section handles errors and error messages.

In terms of items to key in on to create a Snort signature we have:

- The attacking system sends a 0x0065 ENIP command to the controller to elicit a session ID response
- These packets are immediately preceded by the TCP connection setup
- The Session ID changes upon each subsequent connection and is of little value
- Some of these fields may cause false positives in the signature and should be ignored. The Session ID capture mechanism does not contain the actual attack packet containing the CIP data which sets the S2:5/3 fault bit to on

Once the Session ID is obtained, the attack can connect to the controller to obtain the required Connection ID. The next series of packets elicit a response from the controller which allows the capture the Connection ID (note, this is heavily truncated due to the sheer number of fields in ENIP and CIP headers):

	IP Header		ENIP	ENIP Header		CIP		CIP Connection Manager
Packet	Src IP	Dst IP	Session	Command	Session Handle	Service	Request Path	0->T, T->0
7	Metasploit	Micrologix 1400	0x73E5CCAA Send RR Data	0x006f Send RR Data	0x73E5CCAA	Unknown Service 0x54 (Request)	Connection Manager (0x01)	0x80000015 0x80FE0014
8	Micrologix 1400	Metasploit	0x73E5CCAA Send RR Data	0x006f Send RR Data	0x73E5CCAA	Unknown Service 0x54 (Response)	Connection Manager (0x01)	0xAACD2C6F 0x80FE0014

# 3. Using Session ID, Capture Connection ID

Table 3: Using Session ID, request Connection ID and capture

The section of code in the module which creates these packets and captures the Connection ID are in the section called *reqconnection(sessionid)*. As we can see, the *sessionid* variable is passed to this function as it is required to build a packet which will elicit the response required.

The code (truncated) is as follows:

def reqconnection(sessionid)

packet = ""

packet += "\x6f\x00" # SEND\_RR\_DATA (2 bytes)

print\_status("Got connection id: 0x"+connection\_id.to\_s(16))

An analysis of the code shows that the packet is a Send RR Data request in the command field of the ENIP header. The controller responds with a similar message that contains the 0->T network connection ID. The response packet is parsed, looking at bytes 44-48, offset from the start of the ENIP header which is directly after the TCP header we will find the Connection ID which will be stored in the *connection\_id* variable to be used in the attack packet generation. In the example the Connection ID was 0xAACD2C6F. Note that the packet hex in Wireshark that this value is stored in little Endian.

In terms of items to key in on to create a Snort signature we have:

- The static T->0 value as set 0x80FE0014 (note that in packet creation this is backwards due to the Endian-ness of the field)
- The Session ID and controller Connection ID's are both variable and subject to change and are therefore of limited value in a signature
- These packets are immediately preceded by the ENIP registration packets

Once the attacker has the Session and Connection IDs it is possible to forge the attack pack which sets the S2:5/3 bit and implements the fault error on the controller. The attack packet appears as follows (truncated):

#### 4. Send Attack

	IP Header		ENIP	ENIP Header			CIP	CIP Class Generic
Packet	Src IP	Dst IP	Session	Command	Length	Connection ID	Service	Data
							Unknown	
			0x73E5CCAA	0x0070			Service	07 4d 00 3d 09 a9 0a
		Micrologix	Send Unit	Send Unit			0x4b	0f 00 68 dd ab 02 02 84
9	Metasploit	1400	Data	Data	49	0xAACD2C6F	(Request)	05 00 08 00 08 00

Table 4: Attack packet attributes

The code that generates this relies upon building a packet in two part (*payload1* and *payload2*) while also injecting the Session and Connection ID gathered in packets 5 and 8. The code that generates the attack packet is as follows:

def forgepacket(sessionid, connectionid, payload1, payload2)

packet = ""
packet = "\x70\x00" # command: SEND\_UNIT\_DATA (4 bytes)
packet += "\x31\x00" # length (4 bytes)
packet += [sessionid].pack("N") # session identifier (4 bytes) \*\*our session ID was 0x73E5CCAA in this case
packet += payload1 #payload1 part
packet += [connectionid].pack("N") # connection identifier (4 bytes)
\*\*our session ID was 0x73E5CCAA in this case
packet += payload2 #payload2 part
begin

sock.put(packet)

....

This code combines all of the elements into the final attack packet. Payload 1 is somewhat uninteresting in its options as it appears to simply include necessary data elements and fields as required by the protocol. Payload2 presents a more interesting set of data, including one which is not seen in the packets until now. The truncated code we examine next is as follows:

payload2 += "\xb1\x00" # connected data item payload2 += "\x1d\x00" # length payload2 += "\x7d\x14" # connection id payload2 += "\x4b" # service payload2 += "\x02" # request path size payload2 += "\x20\x67\x24\x01" # request path payload2 += "\x07\x4d\x00\x3d\x09\xa9\x0a\x0f\x00\x68" # cip class generic payload2 += "\xdd\xab\x02\x02\x84\x05\x00\x08\x00\x08\x00" # cip class generic

The exploit appears to generate data which is located in the Data field of the CIP Generic Class section of the packet which will be sent to the controller. To examine other "normal" network traffic between the controller and the RSMicroLogix application, data during normal run operations as well as download operations was captured to examine this contents of the Data field using Wireshark. The following screenshot depicts legitimate Run operations when connected to the RSMicroLogix application:

	ait ENIE	traffic (	can no	an - Wire	schark	_			_	-				
File	Edit	View	Go	anture	Analyze	Statistics	Teleph	ony Tools	Help					
The	Luit	VIEW	00 1	aprure	Analyze	Junistics	reiepii	011 <u>y</u> <u>1</u> 001s	Tielb					
					* 2		( <del>(</del>	🗞 🔄 🖗	₽		⊕ Q	0	¥ 🗹 🍢	*
Filter	:								▼ E	xpression	Clear A	Apply		
No.	Ti	me	:	Source			Destinatio	n	1	Protocol		Info		
	363 4.	.15061	19	10.0.0	.125		10.0.0	.135		CIP		Unknown	Service	(0x4b)
	364 4.	15473	30	10.0.0	.135		10.0.0	.125	(	CIP		Success		
	365 4.	.17865	55	10.0.0	.125		10.0.0	.135		CIP		Unknown	Service	(0x4b)
	366 4.	.18465	58 :	10.0.0	.135		10.0.0	.125	(	CIP		Success		
	367 4.	.21865	57	10.0.0	.125		10.0.0	.135	(	CIP		Unknown	Service	(0x4b)
	368 4.	. 22474	40	10.0.0	.135		10.0.0	.125	(	CIP		Success		
	369 4.	. 24006	64	10.0.0	.125		10.0.0	.135		CIP		Unknown	Service	(0x4b)
	370 4.	. 24481	19	10.0.0	.135		10.0.0	.125		CIP		Success		
	371 4.	. 27857	72	10.0.0	.125		10.0.0	.135		CIP		Unknown	Service	(0x4b)
	372 4.	.28466	61	10.0.0	.135		10.0.0	.125		CIP		Success		
	3734.	. 31063	36	10.0.0	.125		10.0.0	.135		CIP		Unknown	Service	(0x4b)
	374 4.	. 31479	97	10.0.0	.135		10.0.0	.125		CIP		Success		
4	275 /	34263	28	10 0 0	125		10 0 0	125		стр		Unknown	Service	(0v4h)
۹ 📃														
± Fr	ame 3	363: 1	L12 b	oytes o	on wire	(896 b	its),	112 byte	s capt	tured (8	96 bits	5)		
+ E1	therne	et II,	, Sro	:: 00:7	3:63:6	L:64:61	(00:7	3:63:61:	64:61)	), Dst:	Rockwe	11_a4:01:9	96 (00:1d	1:9c:a4:01:9
+ Ir	nterne	et Pro	otoco	ol, sro	:: 10.0	0.125	(10.0.	0.125),	Dst: 1	10.0.0.1	35 (10	.0.0.135)		
± Tr	ansmi	ission	n Cor	itrol F	rotoco	l, Src	Port:	49591 (4	9591)	, Dst Po	rt: Eth	herNet/IP-	2 (44818	), Seq: 592
± Et	therNe	et/IP	(Ind	lustria	al Prote	ocol),	Sessio	n: 0x2D5	5107F	, Send R	R Data			
E CO	ommon	Indus	stria	al Prot	:0001									
+	Servi	ice: U	Jnkno	wn Ser	vice (	0x4b) (	Reques	t)						
	Reque	est Pa	ath S	51ze: 2	(word	5)								
-	Reque	est Pa	ath:	Class:	0x6/,	Instan	ce: 0x	01						
	■ 8-E	Bit Lo	ogica	al Clas	s Segm	ent (Ox	20)							
		lass:	: Unk	(nown (	(0x67)									
	<b>⊟</b> 8-E	31t Lo	ogica	I Inst	ance S	egment	(0x24)							
	1	Instar	nce:	0x01										
E C1		ass Ge	eneri	c										
-	Comma	and Sp	pecif	ic Dat	a									
	Dat	ta: 07	74000	DA0363	32006000	200003								

Figure 6: Wireshark capture between controller and RSMicroLogix application

In addition, a packet capture of a download of new code to the controller was examined which is depicted in the screenshot below:

10.0.0.135	10.0.0.150	CIP	Success
10.0.0.150	10.0.0.135	TCP	51373 > E
10.0.0.150	10.0.0.135	CIP	Unknown S
10.0.0.135	10.0.0.150	CIP	Success
10.0.0.150	10.0.0.135	CIP	Unknown S
10.0.0.135	10.0.0.150	CIP	Success
10.0.0.150	10.0.0.135	CIP	Unknown S
10.0.0.135	10.0.0.150	CIP	Success
	40.0.435		
: 00:73:63:61: 00:73:63:61:64 	<pre>ba:61 (00:73:63:61:64 :61 (00:73:63:61:64:(  = IG bit: Ind'  = LG bit: Glot (00:1d:9c:a4:01:96  = IG bit: Ind'  = LG bit: Glot 0.0.135 (10.0.0.135) ol, Src Port: EtherNet tocol), Session: 0x9</pre>	<pre>ital) fill fill fill fill fill fill fill fi</pre>	ess (unicast) address (factory ess (unicast) address (factory 0.150 (10.0.0.150) L8), Dst Port: 513 d Unit Data
Send Unit Data 9 andle: 0x95262 uccess (0x0000 ntext: 0000000 0x0000000 cific Data	(0x0070) c31 0000) 000000000		
	10.0.0.150 10.0.0.150 10.0.0.150 10.0.0.150 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.0.135 10.0.0.135 10.0.0.0.135 10.0.0.0.135 10.0.0.0.135 10.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.	10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.150 10.0.0.150 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.135 10.0.0.150 10.0.13616164:61 (00:73:63:61:64 10:73:63:61:64:61 (00:73:63:61:64 10:73:63:61:64:61 (00:73:63:61:64 10:073:63:61:64:61 (00:73:60	10.0.0.150       10.0.0.135       TCP         10.0.0.150       10.0.0.135       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.150       10.0.0.135       CIP         10.0.0.150       10.0.0.135       CIP         10.0.0.150       10.0.0.135       CIP         10.0.0.150       10.0.0.135       CIP         10.0.0.135       10.0.0.135       CIP         10.0.0.135       10.0.0.135       CIP         10.0.0.135       10.0.0.135       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.135       CIP         10.0.0.135       10.0.0.135       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.150       CIP         10.0.0.135       10.0.0.136       CIO:73:63:61:64:61         10.73:63:61:64:61       (00:73:63:61:64:61)          10.0.0.139       Col:14:9C:a4:01:96       Rockwell_a4:01:96         Rockwell_a4:01:96       (00:1d:9c:a4:01:96)      <

Figure 7: Wireshark capture between controller and RSMicroLogix application w/download

Also worth noting is the fact that the Data field length in normal traffic between the controller and application is variable, ranging from 13 to 36 bytes in length. The attack packet uses a static length of 21 bytes due to the construction of the Data field section by the exploit.

In terms of items to key in on to create a Snort signature we have:

- The attack packet contains a concatenation of payload1, payload2, the Session ID, and the Connection ID
- The Data field under the CIP Class Generic section appears to hold the data which flips the S2:5/3 bit to on, causing the logical fault condition on the controller
- The Data field is static in this attack at 21 bytes in length as well as in content which is known by examining either the packet capture or the exploit code itself
- The attack packet has a ENIP header length of 49 bytes as it is forged by Metasploit

# Snort Rule – Round 1

Based on the information gathered to this point it is possible to write a generic Snort rule which will alert on the attack traffic. Although it is a better practice to write the rule to catch the vulnerability, and not the exploit, given that this attack is not "interactive" in the normal sense we are stuck writing the rule to catch the exploit on the wire.

Here is the Snort rule which detects the flow, 44818/TCP port usage (note: it is set to alert on the IP of the controller in the lab only), the flags in the TCP header, and the content utilizing offset and depth:

alert tcp any any -> 10.0.0.135/32 44818 (msg: "Metasploit Cybati Allen-Bradley MicroLogix Major Fault Error detected!"; flow:established; flags:AP; content: "|70 00 31 00|"; offset:0; depth:4; content: "|4B 02 20 67 24 01 07 4D 00 3D 09 A9 0A 0F 00 68 DD AB 02 02 84 05 00 08 00 08 00 |"; offset:46; depth 27; sid:9999999; rev:1;)

This rule has the following attributes:

- It detects traffic flow from any IP, any source port to the controller destined for port 44818
- It only alerts on established connected, as this attack relies on a TCP connection in order to get the Session and Connection IDs that are required for attack
- It only alerts on a packet with the ACK and PSH flags set, as those are the flags set in the attack packet
- It alerts if there is a content match at the beginning of the ENIP header (offset:0) if the first 4 bytes are 0x0070 followed by 0x0031 which is the command to Send Unit Data followed by a header length of 49 bytes; AND
- The content in the CIP Generic Class section (offset:46), including the Data field, are a match on content
- SID, Msg, and Rev are all generic settings used for testing the alert

#### Snort Rule - Round 1 Testing

The lab systems, as shown in the Lab Setup section, were used to run the attack and get the controller into a fault state. Snort was loaded with the above rule under local.rules and was listening to all traffic on the network and Sguil was used to examine the alerts received by Snort. The rule successfully alerts each time the exploit is run against the system as shown in the screenshot below (note extensive testing of the rules here as they were built up over time):

							local.rules				<b>•</b> –	• ×
<u>F</u> ile	<u>E</u> dit	<u>S</u> earch	Optio	ns	<u>H</u> elp							
		W	arning,	you	are u	sing	the root ac	count, you	ı may har	m your system.		
alert	t tc	p any	any	->	10.	0.0	. 135/32	44818	(msg:	"Metasploit	Cybat	i /

Figure 8. Alert added to local.rules

					S	GUIL-	0.8.0	- Co	nnect	ed To l	ocalhos	st						Ŷ	- 8	×
<u>F</u> ile	Query	<u>R</u> ep	orts Sour	nd: Off	Serv	erNam	e: loc	alhos	t Use	erName	student	Userl	D: 2			201	3-05-	18 16:0	)9:29 G	МТ
Re	ealTime B	Event	s Escalate	d Even	ts															
s		T	Sensor	Ale	ert ID	2	Da	ate/Tir	ne e:E4:0	Sro	IP		SPort	D	st IP	20			Port	
R	T	1	student-o	:	9.1 9.2	20	013-05	5-17 0	6:54:0	is 10. IS 10.	0.10.1		52301	10	).0.0. ).0.1(	30 0.2		4	45	
R	T	4	student-o student-o	1	9.3 8.1	20	013-05 013-05	5-17 0 5-18 0	7:06:5 1:31:3	3 10. 7 10.	0.0.30		52315 49902	10	).0.1( ).0.1(	0.2 0.2		8	8 3	
R	T 5	33	student-o	1	8.8	20	013-05	5-18 0	1:32:2	3 10.	0.0.130		51196	10	0.0.0.	135		4	4818	
	P Resol	ution	Agent St	tatus	Snor	t Statis	tics		show i	Packet	Data $\checkmark$	Snow P	Kule (1/1919 (m	nea: '	Moto	enlo	it Cv	hati		
Sro	Reverse : IP:	DNS	S 🔽 Enable	Extern	al DN	15		aren	IP	So 10.0.0.	urce IP 130	10.0	Dest IP	isy.	Ver 4	HL 5	TOS 0	len 40	ID 45583	lag
Sro Ds Ds	Src Name: Dst IP: Dst Name: Whois Query:   None  Src IP  Dst IP							т	СР	Source Port 51196	Dest F Port 1 44818 .	U A R R R C 1 0 G K	( P R S P S S Y I ( H T N N ( )	F I N X 357	Seq :	# 9108	A 8293	ck # 71225	Offse	t te C
	iois Quei	y	None	SICIP		JSUIP		D	ATA	None.										Ę
	Menu					[Ter	mina	al - ro	ot@		[SGUIL-	-0.8.0 -	Co		GUI	1-0.1	B.0 -	Con		െ

Figure 9. Sguil showing alerts (533 at this point) for alerts on the rule as described above in this section

Snort was left in a listening mode while further "non-exploit" traffic was sent between the controller and RSMicroLogix application (i.e. connect, change mode to run/program, download new code, etc.). No false-positive alerts were witnessed.

#### Snort Rule – Round 2

Keying in on the Data field, further examination of the values in this field may prove to be of interest in writing the Snort rule or creating a tighter version of it to further limit false positive alerts. To test which bytes of the Data field data affect the exploit, that individual bytes in the Data field were "fuzzed" byteby-byte and the exploit re-run to determine if the fault condition would still be induced by the attack. The table below indicates which bytes in the Data field, when changed, either fail to induce the fault or the attack operates as designed:

Payload 2 - CIP generic class data generation section of the exploit						
Position	Value	Exploit remains functional after change	Byte offset in packet from ENIP header			
1	x07	YES	52			
2	x4d	YES	53			
3	x00	YES	54			
4	x3d	YES	55			
5	x09	YES	56			
6	xa9	YES	57			
7	x0a	YES	58			
8	x0f	NO	59			
9	x00	YES	60			
10	x68	YES	61			
11	xdd	YES	62			
12	xab	NO	63			
13	x02	NO	64			
14	x02	NO	65			
15	x84	NO	66			
16	x05	NO	67			
17	x00	NO	68			
18	x08	NO	69			
19	x00	YES	70			
20	x08	YES	71			
21	x00	YES	72			

Table 5. Table outlining the success of the attack when specific bytes of the data in the CIP Data field are modified

From the table above it appears that the byte values of more than half of the Data section appear to not affect the attack's success. The Snort alert rule was refined based on the information above and further tested.

The new Snort rule which was tested was as follows (note the sid was changed to determine when the "new" or refined rule was hit during testing):

alert tcp any any -> 10.0.0.135/32 44818 (msg: "Metasploit Cybati Allen-Bradley MicroLogix Major Fault Error detected!"; flow:established; flags:AP; content: "|OF|"; offset:59; depth:1; content: "|AB 02 02 84 05 00 08|"; offset:63; depth 7; sid:9999998; rev:1;)

The rule above was added to local.rules, Snort restarted, and Sguil opened again. The exploit was run once again and the new alert appeared as highlighted in the screenshot below.

e <u>Q</u> u	ery <u>R</u> e	eports Sound	i: Off Server	Name: localhost UserN	ame: <mark>student</mark> U	IserID: 2	2013-05	5-18 23:12:42
RealTir	me Eve	nts Escalated	I Events					
ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort
RT	1	student-o	9.1	2013-05-17 06:54:09	10.0.10.1	2600	10.0.0.30	9997
RT	1	student-o	9.2	2013-05-17 06:54:43	10.0.0.30	52301	10.0.10.2	445
RT	4	student-o	9.3	2013-05-17 07:06:53	10.0.0.30	52315	10.0.10.2	88
RT	1	student-o	8.1	2013-05-18 01:31:37	10.0.0.30	49902	10.0.10.2	53
RT	537	student-o	8.8	2013-05-18 01:32:23	10.0.0.130	51196	10.0.0.135	44818
RT	8	student-o	8.539	2013-05-18 16:26:10	10.0.0.130	55315	10.0.0.135	44818
RT	1	student-o	9.8	2013-05-18 18:59:57	10.0.0.2	24192	10.0.0.125	3389
RT	1	student-o	8.547	2013-05-18 23:12:14	10.0.0.130	52740	10.0.0.135	44818

IP Resolution Agent Status Sport Statistics	Show Packet Data Show Rule						
	alert tcp a	alert tcp any any -> 10.0.0.135/32 44818 (msg: "Metasploit Cybati					
Reverse DNS  ✓ Enable External DNS	IP	Source IP	Dest IP	Ver HL T	OS len	ID lag	
Src IP:		10.0.0.130	10.0.0.135	4 5 0	113	9205 2	
Src Name:	тср	Source Dest R R Port Port 10	UAPRSF RCSSYI GKHTNN	Seq #	Ack #	Offset le	
Whois Query:  None  Src IP  Dst IP		52740 44818	. X X 2	005041586	38391383	5 0	
	DATA	70 00 31 00 D3	3 60 B1 1E 00 A AA AA AA AA	00 00 00 00 00 00	00 00 A1 AA	00 00 I	
		6	oorob Dookot Do	deed 0.1	low G Tr		

Figure 10. Sguil window with new alert added to local.rules and exploit run against the controller

#### Conclusion

The rules as written in either case should function appropriately as neither rule alerted upon normal controller to application network traffic, and only alerted upon running the exploit and creating the logical fault condition on the controller. Obviously the second rule is tighter as it only keys in on specific bytes in the Data field which cause the fault condition to occur. If more time to devote to this work was available it would be recommended that the CIP Data field and the various byte elements be examined further. While the elements of the CIP Data field which result in the setting of the S2:5/3 bit is known, the structure of this field is not known. Research on this topic has not produced a succinct definition of the Data field which could be applied to the attack being examined.

Although, many of the documents which define ENIP and CIP were examined to determine what the values in the CIP Command Specific Data section were nothing conclusive was determined. However,, it appears the byes in the Data field are related to the following table:

Structure	Field	Bytes	Type	Description
Packet Number	Sequence Count	2	UINT	NOT IN UNCONNECTED MSG; requestor
Message Router	Service Code	1	USINT	0x4B Execute PCCC service request code
Service Request	Size of Req_Path	1	USINT	0x02 Path Size in words
	Request_Path	size	Array byte	EPATH 20,67 (class, PCCC); 24,01 (Instance 1)
MR Service	Execute_PCCC	1	USINT	Lenght of Requestor ID (in bytes)
Request Data	Requestor ID			(vendor + s/n + other + 1)
		2	UINT	CIP Vendor ID of requestor
		4	UDINT	CIP serial number
		var	Array byte	"Other" - may not be present
	Execute_PCCC	1	USINT	CMD - Command byte; typically 0x0F or 0x06
	PCCC Command	1	USINT	STS - 0x00 in request
		2	UINT	TNSW - Same value in request and response
		1	USINT	FNC - not used for all CMDs
		var	Array byte	PCCC CMD/FNC specific data 244 max

7-Jun-01

Copyright 2001©Rockwell Automation Technology Transfer and Services p22 of 43

If this is accurate, then the values we are keying in on in our Snort rules are:

0x0F – CMD byte

0xAB - FNC

0x02 0x02 0x84 0x05 0x00 0x08 - PCCC CMD/FNC specific data

One final note: research did turn up some proposed modifications to MicroLogix controller, specifically the 1200 and 1500 series controllers where the S2:5/3 bit will only be "clearable" through communication messages but not writable to mitigate the attack described in this submission. These changes were slated for firmware updates released in March 2013.