DoS Exploitation of Allen-Bradley's Legacy Protocol through Fuzz Testing*

Francisco Tacliad Thuy D. Nguyen Mark Gondree

*The views expressed in this material are those of the authors and do not reflect the official policy or position of the Naval Postgraduate School, the Department of Defense, or the U.S. Government.

December 5, 2017

Introduction

- EtherNet/IP is a TCP/IP-based industrial protocol commonly used in industrial control systems (ICS)
- Using a custom Scapy-based fuzzer, we uncover a previously unreported denial-of-service (DoS) vulnerability in the Ethernet/IP implementation of the Rockwell Automation/Allen-Bradley MicroLogix 1100 PLC
- ICS-CERT recently announces this vulnerability in the security advisory ICSA-17-138-03

Introduction

- Modern industrial network protocols have evolved from serial-based fieldbus protocols to TCP/IP-based protocols that are transported over standard Ethernet links
- Common Industrial Protocol (CIP) [21] and Ethernet/Industrial Protocol (EtherNet/IP) [22] are two well-known Open DeviceNet Vendors Association (ODVA) TCP/IP-based industrial protocols used by large number of industrial automation vendors
- Rockwell Automation/Allen-Bradley (RA/AB) PLCs (e.g., ControlLogix and MicroLogix) implement these protocols

Introduction

- Fuzz testing, or fuzzing, is a penetration testing technique to verify the robustness of target software in handling invalid, malformed, or unexpected input data
- Fuzzing the implementations of control network protocols is an important step towards developing more secure industrial control systems
- Little information has been made publicly available on the vulnerabilities of the EtherNet/IP software used in commercial PLCs
- To examine the robustness of the EtherNet/IP implemenation of select RA/AB devices, we create a fuzz testing tool (ENIP Fuzz) using Scapy, a Python module used for packet parsing and crafting [19]

Introduction

- A Scapy-based fuzzer for exploiting EtherNet/IP security vulnerabilities
- Remote fault detection strategy
- Deficiency in MicroLogix's handling of the Programmable Controller Communication Commands (PCCC) protocol
- Preliminary exploration of potential cross-generational vulnerabilities

EtherNet/IP Protocols

Common Industrial Protocol (CIP)

- `objects`: particular component within a product
- `class`: a set of objects of the same component
- `object instance`: actual representation of particular object
- `instance`: class or object share same attributes, but has own unique values [21]

EtherNet/IP

- Allow CIP communications to be transported over standard Ethernet
- TCP and UDP over port 44818
- Implicit messaging enables exchange of scheduled, time-critical control data [22]
- Explicit messaging provides general request reply/reply communication [22]

EtherNet/IP Protocols

Programmable Controller Communication Commands (PCCC)

- Provides legacy support for older RA/AB PLCs, e.g., PLC5 and SLC500 [7]
- Used with EtherNet/IP, encapsulated in CIP
- Encapsulation is accomplished through "Execute_PCCC" CIP service (service code = 0x4B)
- Each message contains command code and function code

Fuzzing Methodologies

Mutation-based fuzzers

- Apply transformations (mutations) on existing data samples to create test cases
- Brute force testing

Generation-based fuzzers

- Test cases employ rules defining a grammar-based specification for inputs
- Requires up-front understanding of specification or source code

ICS Protocol Fuzzers

Name	Type	Protocol	Availability
Aegis Fuzzer [2, 3]	custom	DNP3, Modbus	commercially licensed, early version open-source
Beyond Security's beSTORM [5]	framework	several, including DNP3	commercially licensed
blackPeer [10]	framework	several, including Modbus	NA
Codenomicon's Defensics [11]	framework	several, including CIP, EtherNet/IP, Modbus, OPC UA Server, Profinet, Scada GOOSE	commercially licensed
ICCP Fuzzer [13]	custom	ICCP	NA
LZFuzz [9]	framework	several, including SNMP [20]	NA
MTF [25]	custom	Modbus	NA
OPC-MFuzzer [26]	custom	OPC, DCOM, RPC [18]	NA
OPC Server Fuzzer [15]	custom	OPC Server	NA
Peach [16]	framework	several, including Modbus, BACNet, DNP3, OPC [16, 26]	open-source
ProFuzz [14]	custom	Profinet	open-source
scada-tools [24, 23]	custom	Profinet	open-source
Sulley [17]	framework	several, including Modbus, DNP3, TPKT, COPT [12]	open-source
Wuldtech's Achilles [1]	custom	several, including EtherNet/IP, Foundation Fieldbus, MMS, Modbus, OPC UA, Profinet, DNP3, MMS, SES-92	commercially licensed

Design and Implementation

Fuzzers operate under two basic assumptions:

- Faults in a target application can be triggered through input controlled by the user
- The execution of a faulty portion of an application will result in some behavioral manifestation (e.g., bricking the device or producing unexpected output)

Implementation of Support Library

- Library uses Scapy, a Python module used for packet crafting and manipulation
- Library conforms to EtherNet/IP specifications [22, 21]
- ENIP Fuzz is complete in its support of EtherNet/IP and one fourth of CIP specification
- EtherNet/IP traffic characterized from ICS lab environment, which included the AB/RA MicroLogix 1100 and ControlLogix 5570

Implementation of Fuzzer

EtherNet/IP Register Session Request

- Used for establishing a session between an originator and a target
- Originator sends Register Session Request on port 0xAF12, the target shall assign and reply with a Session Handle [22]

CIP NOP Request

- CIP common service request that generates a No Operation Response [21, §A-4.17]
- Receiver does not execute any other internal action

Implementation of Fuzzer

Execute PCCC Service

- PCCC is a vendor specific application layer protocol used for communication between certain RA/AB processors
- Used primarily to "ease communication between legacy networks and the new CIP networks" [6, p. 7.17]
- The Protected Typed Logical Read with Three Address Fields command is the specific Execute PCCC Service function chosen for fuzzing; function is used to read data from a logical address [6, p. 7.17].

Fault Monitoring

Liveness Check

- Remote analysis to determine crashes occurred
- TCP RST Flag for indicating target device has crashed [20]
- Socket timeout, reset, or close; failure in reopening a closed socket; and failure in opening a new socket [25]

Unexpected responses

- Filter for responses outside of specification

Performance degradation

- Malformed packets impacting timely delivery of responses may be considered soft failure
- Records captured during fuzzing are compared to baseline and analyzed for irregularities in response times

Test Environment

SUT

- MicroLogix 1100

Fuzzer

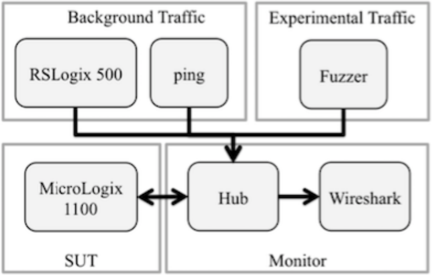
- Kali 2.0 VM with the fuzzer

Background traffic generators

- Windows 7 Virtual Machine with RSLinx
- Kali 2.0 VM with the Ping Utility

Monitor

- Mac OS X running Wireshark



Experimentation

- A liveness check is performed using the Ping utility to determine that the target is still responsive
- Monitor the latency in responses to both ICMP Echo requests and EtherNet/IP requests made by the RSLinx
- SUT is also monitored for unexpected responses, i.e., responses outside the EtherNet/IP specification or otherwise incorrect (e.g., responses that contain erroneous data).

Results and Analysis

- Three metrics were used for analysis: the deltas between ICMP Echo requests from Ping, List Identity requests from RSLinx, and the response from the service request being fuzzed
- SUT interacts with the traffic generators during "warm-up period," fuzzer sends either correctly formed packets (during baseline) or malformed packets (during testing) for a period of approximately 20 minutes
- Wireshark packet capture of the fuzzing session is then truncated into a 10 minute window, after which each of the metrics is analyzed
- Each delta is calculated by taking the difference between the timestamp of the response and the request.

Results and Analysis

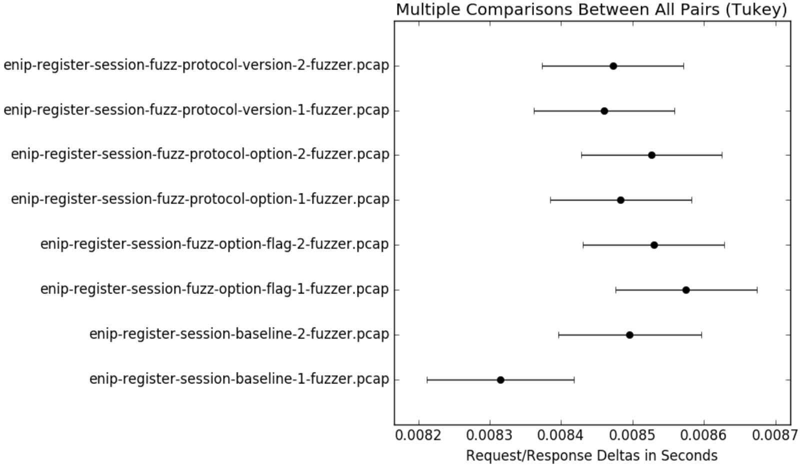
- Three baseline measurements and fourteen trials were performed
- Each baseline and trial was repeated twice

Trial Name	Field Fuzzed
enip-register-session-baseline	NA
enip-register-session-fuzz-protocol-version	Version
enip-register-session-fuzz-option-flag	Options
enip-register-session-fuzz-protocol-option	Version,Options
cip-nop-baseline	NA
cip-nop-fuzz-class	Class
cip-nop-fuzz-instance	Instance
cip-nop-fuzz-class-instance	Class,Instance
pccc-exec-baseline	NA
pccc-exec-fuzz-byte	Byte Size
pccc-exec-fuzz-file-no	File Number
pccc-exec-fuzz-file-type	File Type
pccc-exec-fuzz-element	Element No.
pccc-exec-fuzz-all	File No., File Type, Element No.

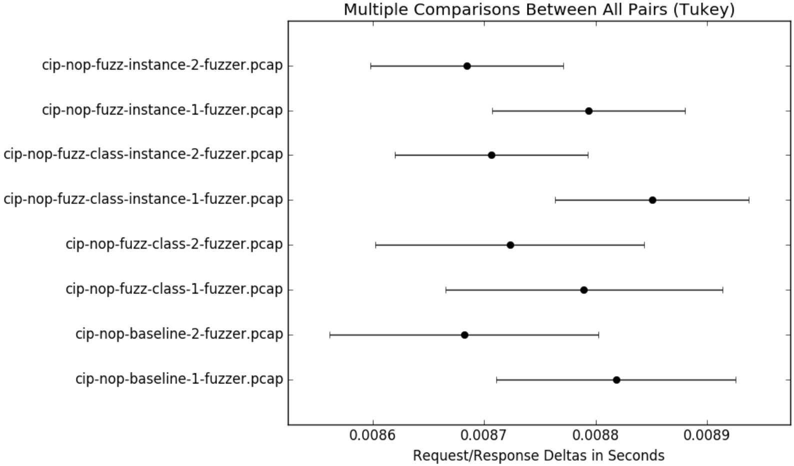
Response Time Analysis

- Deltas in response times from ICMP Echo requests and List Identity requests may not be meaningful metrics
- Using Tukey's Honest Significant Difference (HSD) test there is no significant difference in response times when fuzzing compared to when sending non-malformed traffic

Response Time Analysis



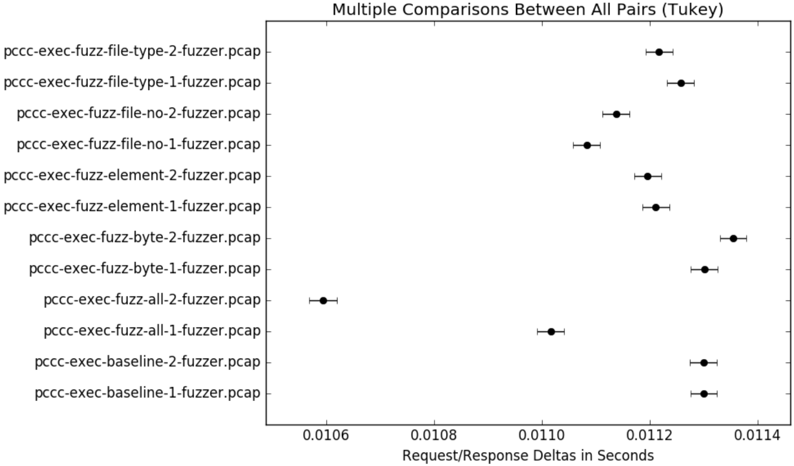
Response Time Analysis



Response Time Analysis

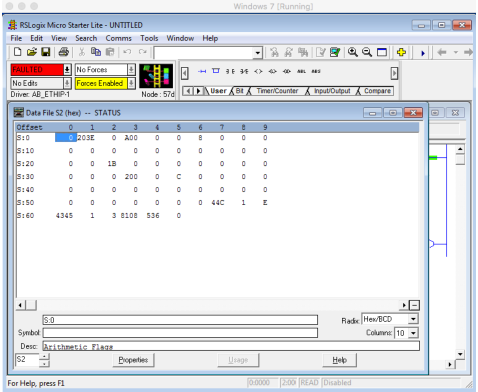
- Tests against Execute PCCC Service shows some sensitivity with performance metrics
- More testing warranted to claim fuzzed inputs were responsible for performance degradation

Response Time Analysis



Denial-of-Service Fault

- When fuzzing the Execute PCCC Service, we discover a previously unreported DoS vulnerability
- To clear fault, device must be power-cycled and reset using RSLogix Clear Major Fault utility
- SUT used to test fault condition is a MicroLogix 1100 PLC (1763-L16BWA Series B, FRN 14)



Denial-of-Service Fault

- To exploit the vulnerability, the attacker sends a single Execute PCCC Service - Protected Typed Logical Read with Three Address Fields packet with a File Number of 0x02–0x08 and File Type 0x48 or 0x47. Any combination of File Number 0x02–0x08 and File Type 0x48 or 0x47 will trigger a Major Error (0x08)
- Data files store status and data information associated with instructions used in ladder subroutines [8, p. 40–41]

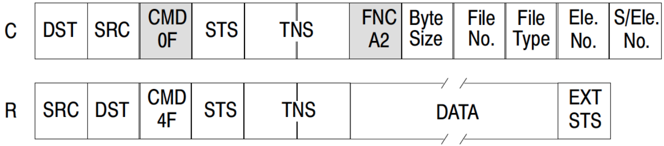
```
###[ ENIP TCP ]###
           Command   = Send Unit Data (0x0070)
           Length    = 45
           Session_Handle= 0x6077596d
           Status    = Success
           Sender_Context= 0
           Options   = 0
###[ Send Unit Data ]###
              Interface_Handle= 0
              Timeout   = 20
###[ ENIP_CommonPacketFormat ]###
                 Item_Count= 2
                 \Items     \
                  |###[ Common Packet Format Item ]###
                  |  Address_Data_Item= Connection-Based (0x00A1)
                  |  Address_Length= 4
                  |  Connection_Identifier= 0x6d596902
                  |###[ Common Packet Format Item ]###
                  |  Address_Data_Item= Connected Transport Packet (0x00B1)
                  |  Data_Length= 25
                  |  Sequence_Number= 0x1
###[ Common_Industrial_Protocol ]###
                    Request_Response= Request
                    Common_Service= Execute_PCCC_Service
                    Request_Path_Size= 2
                    \Words     \
                     |###[ CIP Request Path ]###
                     |  Path_Segment_Type= Logical Segment
                     |  Logical_Segment_Type= Class ID
                     |  Logical_Segment_Format= 8-bit logical address
                     |  Class     = 0x67
                     |###[ CIP Request Path ]###
                     |  Path_Segment_Type= Logical Segment
                     |  Logical_Segment_Type= Instance ID
                     |  Logical_Segment_Format= 8-bit logical address
                     |  Eight_bit_Instance= 0x1
###[ CIP Execute PCCC Service Request ]###
                       Length_of_Requestor_ID= 7
                       CIP_Vendor_ID_of_Requestor= Rockwell Software, Inc.
                       CIP_Serial_Number= 90180339
                       CMD       = 0x0F
                       Status    = 0x0
                       Transaction_Word= 2
                       Function  = Protected Typed Logical Read Three Address Fields
                       Byte_Size = 0x0
                       File_No   = 0x5
                       File_Type = 0x47
```

ControlLogix Experiment

- We speculate that the same PCCC vulnerability could potentially exist in newer RA/AB PLC models
- Same PCCC stress tests on the ControlLogix 5570 did not cause expected DoS fault
- Experiment yields insight into the differences in the way MicroLogix and ControlLogix respond to the Protected Typed Logical Read with Three Address Fields PCCC command

ControlLogix Experiment

- A PCCC reply always has a `status (STS)` byte, and for some commands, an `extended status (EXT STS)` byte
- MicroLogix only returns the STS byte (0x10= "Illegal command or format") whereas ControlLogix returns both STS and EXT STS bytes– STS = 0xF0 ("Error code in the EXT STS byte") and EXT STS = 0x06 ("Address doesn't point to something usable")[6]
- Functional difference indicates that it may be more valuable to fingerprint PLCs using information at the application level



Future Work

- EtherNet/IP support library can be expanded so that is fully compliant with specifications
- Testing TCP and IP layers may expose vulnerabilities in the ENIP/IP implementation
- Explore EtherNet/IP implementations across related products, i.e., products that conforms to the ODVA specification or deemed interoperable with related models
- OpENer is a POSIX-compliant implementation of ENIP that is partially supported by Rockwell Automation [4]

Questions

Francisco Tacliad - francisco.tacliad@gmail.com
Thuy D. Nguyen, Naval Postgraduate School, tdnguyen@nps.edu
Mark Gondree, Sonoma State University, gondree@sonoma.edu

References I

[1] *Achilles Platform*. Accessed: March 25, 2016. n.d. URL: https://www.wurldtech.com/product/achilles.

[2] *Aegis*. Accessed: March 25, 2016. n.d. URL: https://www.automatak.com/aegis/.

[3] Automatak. Accessed March 25, 2016. aegis-opensource. URL: https://github.com/ITI/ICS-Security-Tools/tree/master/protocols.

[4] Rockwell Automation. *Rockwell Automation Sponsors Development of Open-Source Software Stack*. Accessed Sept. 8, 2017. 2009. URL: http://phx.corporate-ir.net/phoenix.zhtml?c=196186&p=irol-newsArticle&ID=1356918.

References II

[5] *BeSTORM software software security testing tool.* Accessed: March 25, 2016. n.d. URL: http://www.beyondsecurity.com/bestorm.html.

[6] Allen Bradley. *DF1 Protocol and Command Set, Reference Manual.* 1770-65.16. Milwaukee, WI, 1996.

[7] Allen Bradley. *Logix5000 Data Access Programming Manual.* 1756-PM020D-EN-P. Milwaukee, WI, 2016.

[8] Allen Bradley. *MicroLogix 1100 Programmable Controller Instruction Set Reference Manual.* 1763-RM001B-EN-P. Milwaukee, WI, 2011.

[9] Sergey Bratus, Axel Hansen, and Anna Shubina. *LZfuzz: a fast compression-based fuzzer for poorly documented protocols.* Tech. rep. TR-2008 634. Hanover, NH: Darmouth College, Sept. 2008.

References III

[10] Eric J Byres, Dan Hoffman, and Nate Kube. "On Shaky Ground–A Study of Security Vulnerabilities in Control Protocols". In: (2006), pp. 782–788.

[11] *Defensics*. Accessed: March 25, 2016. n.d. URL: http://www.codenomicon.com/products/defensics/.

[12] Ganesh Devarajan. "Unraveling SCADA Protocols: Using Sulley Fuzzer". In: Las Vegas, NV, 2007.

[13] Matthew Franz. "ICCP Exposed: Assessing the Attack Surface of the Utility Stack". In: *Proceedings of SCADA Security Scientific Symposium*. Miami, FL, 2007.

[14] Roland Koch. Accessed March 25, 2016. ProFuzz. URL: https://github.com/HSASec/ProFuzz.

[15] Luis Mora. *OPC Security White Paper*. Byres Research. Jan. 2007. URL: https://scadahacker.com/library/Documents/OPC_Security/OPC%20Security%20-%20OPC%20Exposed.pdf.

[16] *Peach Introduction*. Accessed: March 25, 2016. Deja Vu Security, n.d. URL: http://community.peachfuzzer.com/Introduction.html.

[17] Aaron Portnoy, Pedram Amini, and Ryan Sears. Accessed March 25, 2016. sulley. URL: https://github.com/OpenRCE/sulley.

References V

[18] Xiong Qi et al. "OPC-MFuzzer: A Novel Multi-Layers Vulnerability Detection Tool for OPC Protocol Based on Fuzzing Technology". In: *International Journal of Computer and Communication Engineering* 3.4 (July 2014). URL: http://search.proquest.com/docview/1618797232?pq-origsite=gscholar.

[19] *Scapy*. Accessed: Aug. 14, 2016. 2011. URL: http://www.secdev.org/projects/scapy.

[20] Rebecca Shapiro et al. "Identifying Vulnerabilities in SCADA Systems via Fuzz-Testing". In: *International Conference on Critical Infrastructure Protection*. Heidelberg, Germany, 2011, pp. 57–72.

References VI

[21] *The CIP Networks Library Volume 1: Common Industrial Protocol*. 3.22. Open DeviceNet Vendor Association, Inc. Ann Arbor, MI, Apr. 2017.

[22] *The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP*. 1.23. Open DeviceNet Vendor Association, Inc. Ann Arbor, MI, Apr. 2017.

[23] A. Timorin. *Scada deep inside: protocols and security mechanisms*. unpublished.

[24] Alexander Timorin. Accessed March 25, 2016. URL: https://github.com/atimorin/scada-tools.

[25] Artemios G Voyiatzis, Konstantinos Katsigiannis, and Stavros Koubias. "A Modbus/TCP fuzzer for testing internetworked industrial systems". In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation*. Luxembourg City, Luxembourg, 2015, pp. 1–6.

References VII

[26] Ting Wang et al. "Design and Implementation of Fuzzing Technology for OPC Protocol". In: *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Beijing, China, 2013, pp. 424–428.