ARMORE
Applied Resiliency for More Trustworthy Grid Operation
Research Update

Tim Yardley
yardley@illinois.edu
About Me

- Tim Yardley, Associate Director of Technology
- Information Trust Institute, University of Illinois Urbana-Champaign

- Old school hacker, Long time practitioner, Current researcher

- @timyardley, yardley@illinois.edu
UIUC’s Information Trust Institute

Providing World-Wide Excellence in Information Trust and Security

Institute Vision:
Trust in Complex Systems

Institute Personnel:
Core faculty from CS and ECE
90+ faculty, 28 departments, 11 colleges

Primary Research Themes
• Power Grid
• Evaluation
• Data Science
• Systems and Networking

Background
• Since 2004 startup ITI has won $100M+ in research funding
• Solutions for societal and industrial problems
• Major corporate partnerships
• Led by the University of Illinois College of Engineering
Smart Grid Security Efforts @ Illinois

Centers

Trustworthy Cyber Infrastructure for the Power Grid (~$26.3M effort across 10 years)
- Drive the design of a more secure, resilient, and safe electric power infrastructure
- University of Illinois, Washington State, Dartmouth, Arizona State

Smart Grid Subprogram (~$15M effort across 5 years)
- Cybersecurity, Microgrids, DERs, and HANs

Illinois Center for a Smarter Electric Grid (~$5M effort across 5 years)
- Validation of IT and control aspects of the Smart Grid
- Operates facilities equipped with HW/SW to aide in the validation of emerging smart grid systems
- Focus on both power and cyber related issues

Assured Cloud Computing (~$6M effort across 6 years)
- Leveraging trustworthy cloud computing for critical infrastructure

Science of Security Systems (~$8.5M effort across 4 years)
- Resiliency, security, and trust in complex engineered systems

Highlighted Projects

Policy Based Configuration (PBCONF)

Software Defined Networking

Applied Resiliency for More Trustworthy Grid Operation (ARMORE)

Collaborative Defense for T&D Devices Against Attack (CODEF)

Cyber-Physical Modeling and Analysis for a Smart and Resilient Grid

... and many more
Overview
Motivation

- Industrial Control Systems (ICS) protocols lack security protection
- Security bolt-ons are typically implemented via firewalls and VPNs
- Little if any visibility as to what these systems are actually doing
- Any security extensions have a long-tail implementation path (or never at all)
- Deployments are often much more costly than the capital expenditures
What is ARMORE?

• Security appliance that aims to
  • *Increase visibility and awareness on ICS networks*
  • *Augment insecure protocols with security features*
  • *Inspect and (optionally) enforce defined policies*
  • *Minimize deployment costs while creating a feasible adoption path*
How ARMORE Works

• Passive
  • *Span port*

• Transparent
  • *Inline inspection, optional enforcement*

• Encapsulated
  • *Inline inspection, encapsulated transfer with optional encryption, optional enforcement*
What do you get?

- Passive
  - *Network visibility and intelligence*
- Transparent operation
  - *Passive plus…*
  - *Communication endpoints operate without any changes*
  - *Optional policy enforcement*
- Encapsulated
  - *Transparent plus…*
  - *Encapsulation and Encryption*
  - *Security augmentation (access control/filtering)*
  - *Optional policy enforcement*
  - *Fault tolerance and resiliency options*
- Other value adds
  - *Enhanced access control*
  - *Payload inspection*
  - *Data processing and analysis*
ARMORE Conceptual Diagram
In deployment...
System Realization

Work by Steve Granda
ARMORE Software

- OS: Debian Wheezy 7.8 x64
  - Modified 3.12.0 Linux Kernel
- ARMORE Proxy
  - Abstracted middleware encapsulator
- Bro
  - Intrusion Detection System
- NetMap
  - Kernel Module for High Speed Packet I/O
- Management/Configuration
- ZMQ
  - Middleware layer
  - CurveZMQ
    - Authentication and Encryption protocol for ZMQ
Other ARMORE Support

- BrocolliSharp
- Bro-statsd
- Rsyslogd
- Etckeeper
ARMORE Node installation

- Original installation was via a large shell script which compiled and installed software from source.
- Current installation is with our debian repository

- Allows easier dependency checking and updating of individual components.
  - `apt-get install armorenode`
  - `apt-get update armorenode`
Middleware

Work by Chris Drew and Steve Granda
Scope in ARMORE
ARMORE Proxy

- Abstract class for middleware library inclusion
  - ZeroMQ implemented with Curve security
  - DDS stubbed but not implemented
    - Reason: Open source libraries are currently lacking security extensions
- Abstract packet capture interface
  - PCAP
  - Netmap
- Many options for logging
- MAC address translation mode
ZMQ

• Asynchronous messaging library
• Allows many types of communication from intra-process to WAN
• Removes need for message broker
• API values simplicity over functionality
• Encourages user to implement functionality as needed
• Available in over 30 languages on multiple platforms
• Open source
• Very active community provides extensive support for developing and debugging
• Existing documentation provides extensive instruction on various communication patterns
ZMQ - Patterns

• Provides ability to create many communication patterns

• ARMORE is utilizing a dealer/router pattern
ZMQ Dealer/Router Pattern
**DDS vs. ZMQ**

**DDS**
- Commercial Product
- Desired functionality built in
- Steep learning curve
- Sightly more resource heavy
- ~4 languages
- Restricted to pub/sub

**ZMQ**
- Open source
- Some functionality may need to be written
- Easy to learn
- Lightweight
- 30+ languages
- Flexible to multiple patterns
System Administration

Work by Chris Drew
Web API

- Front end connects UI with ARMORE node internals
  - *Read/set configuration*
    - Subsystem status
    - Node topology
  - *Display data for user*
    - Statistics
    - Logs
    - Alerts
- Communicate with back end via JSON messages
- Testing
  - *Janus* - Rest API server
  - *Bottle* - Python Web Framework
Example Endpoints

• armore/config/zmq/5 (NOTE: node id 5)
  
  ```json
  {
      "Encryption": True,
      "Reliability": "Best Effort",
      "Durability": "Transient Local"
  }
  ```

• armore/notifications/bro
  
  ```json
  {"eventId": [{
      12: {
          "time": "7/13/2013 12:45:01",
          "srcNode": "Node_2",
          ...
      },
      58: {
          "time": "9/3/2013 12:45:01",
          "srcNode": "Node_91",
          ...
      }
  }]
  ```
Dynamic and Smart Traffic Analyzer for Smart Grid

Work by Wenyu Ren
Introduction

• What is it?
  • An analyzer that provides dynamic and intelligent analytics for SCADA protocols, increasing visibility into the system behavior

• What is it using?
  • Bro's scripting engine

• What protocols does it support at the moment?
  ✓ DNP3
  ✓ Modbus
  ✓ Extensible to any other protocol
Network Traffic

Traffic Statistics Collector

Traffic Statistics Counter

Pattern-based Identity Recognition

Anomaly Detection Framework

Structure
Traffic Statistics Collector

- Input: network traffic
- Output: two kinds of events
  - `item_seen`: instantaneous, item contains incomplete information of the packet
  - `item_gen`: delayed, item contains complete information of the packet
Traffic Statistics Collector

Network Traffic

Traffic Statistics Collector

item_seen

Traffic Statistics Counter

item_gen

Pattern-based Identity Recognition

Log
### Traffic Statistics Collector

- **Trace:** synthetic Modbus traces

<table>
<thead>
<tr>
<th>Subject</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Packet Interval</td>
<td>6ms 876us</td>
</tr>
<tr>
<td>Average Burst Interval</td>
<td>1s 824ms</td>
</tr>
<tr>
<td>Average Burst Length</td>
<td>32</td>
</tr>
<tr>
<td>Total Valid Time</td>
<td>1h 2min</td>
</tr>
<tr>
<td>Total Packet Number</td>
<td>60227</td>
</tr>
</tbody>
</table>
Traffic Statistics Collector

- 5 level

Sender & Receiver
- Trigger Time $T_{11}$
- Extract Time $T_{12}$

Extractor
- Trigger Time $T_{20}$
- Switching Time $T_{21}$
- Extract Time $T_{22}$

Protocol & Function
- Trigger Time $T_{23}$
- Extract Time $T_{24}$

Target
- Extract Time $T_{31}$

item_seen
- Trigger Time $T_{32}$
- Trigger Delay $T_{40}$

item_gen
- Trigger Time $T_{41}$

item_seen Receive Time
- $T_{13}$

item_seen Receive Time
- $T_{23}$

item_seen Receive Time
- $T_{33}$

item_gen Receive Time
- $T_{42}$

Total Runtime
Traffic Statistics Collector

- 5 level
Traffic Statistics Collector

- 3-4 level

Total Runtime
Traffic Statistics Collector

- 3-4 level

4 level

3 level
Traffic Statistics Collector

- 1-2 level

Total Runtime
Traffic Statistics Collector

- 1-2 level
Traffic Statistics Collector

- Total Runtime

![Graph showing the relationship between Number of Levels and Time (µs)]
Traffic Statistics Counter

- Multi-level Statistics
- Data Structure: Tree of depth 6

<table>
<thead>
<tr>
<th>Level</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sender</td>
</tr>
<tr>
<td>2</td>
<td>Receiver</td>
</tr>
<tr>
<td>3</td>
<td>Protocol</td>
</tr>
<tr>
<td>4</td>
<td>Function</td>
</tr>
<tr>
<td>5</td>
<td>Target</td>
</tr>
</tbody>
</table>

Function Name
Request or Response
Response Ratio (if request)
Response Delay (if request)
Traffic Statistics Counter

- Item process time $T_5$ is calculated per item_seen event. We further add all the item process time according to the same packet to calculate a total item process time per packet $T_5'$. 
Traffic Statistics Counter

- Time flow comparison of the collector when running different programs
Traffic Statistics Counter

- Total item process time per packet with different number of levels

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>8</td>
</tr>
<tr>
<td>Receiver</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>1</td>
</tr>
<tr>
<td>Function</td>
<td>262</td>
</tr>
<tr>
<td>Target</td>
<td>37</td>
</tr>
</tbody>
</table>
Traffic Statistics Counter

- Aggregation time with different number of levels and different aggregation period $T_p$

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>8</td>
</tr>
<tr>
<td>Receiver</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>1</td>
</tr>
<tr>
<td>Function</td>
<td>262</td>
</tr>
<tr>
<td>Target</td>
<td>37</td>
</tr>
</tbody>
</table>

![Aggregation Time Diagram]
Anomaly Detection Framework

- Each logging of traffic statistics will generate a data structure like a B-tree.

Each Node:
- **Count**: Number of events represented by this node which happened in this logging period
- **Children[]**: Pointers to children of this node
Anomaly Detection Framework

- SCADA traffic is periodic
- May vary in short time, but has a pattern over time.
- Construct “normal” tree and use it as a criterion

When to send notice

Tree\textsubscript{i} Tree\textsubscript{i+1} Tree\textsubscript{i+2} ... Tree\textsubscript{i+k} Tree\textsubscript{i+k+1}
Anomaly Detection Framework

- Anomaly detection time with different number of levels and different aggregation period $T_p$

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>8</td>
</tr>
<tr>
<td>Receiver</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>1</td>
</tr>
<tr>
<td>Function</td>
<td>262</td>
</tr>
<tr>
<td>Target</td>
<td>37</td>
</tr>
</tbody>
</table>
Pattern-Based Identity Recognition

item_gen → Pattern-based Identity Recognition → id_recognized

ID Process Time $T_I$
Pattern-Based Identity Recognition

- Time flow comparison of the collector when running different programs
Example Uses of Analytics

• If one can inspect the communications, one can observe patterns and behaviors
  • E.g., DNP3 SBO message, with affirmative response…
    • Probably a relay

• With inspection, one can then enforce
• What’s going on in your network?
• Future planning
• Encryption
• Fault-tolerance
System Testing

Work by Chris Drew and Steve Granda
Physical Test bed Overview
Proxy Testing Procedures

- With the armoreconfig service running on armorenodes we generate traffic with iperf on the blue node to the red node.
  \[
  \text{Iperf} \ -c \ 192.168.2.15 \ -i \ 1
  \]
- In the above example we’ll listen for generated traffic with tcpdump on red node:
  \[
  \text{tcpdump} \ -i \ \text{eth0}
  \]
Testing Procedures

- Once the ARMORE Nodes are configured we use the pkt-gen script to send data on a netmap pipe:
  
  ```
  ./pkt-gen \-i netmap:eth0\{0 \-f rx
  ```

- Anything listening on netmap pipe `eth0\{0` should be able to transparently receive data.
Server Room
SCADA/ICS Testing

• DNP3 and Modbus Protocol Test Harnesses
  • *Will generate typical traffic and verify back and forth connectivity*
  • *Leveraging open stacks for implementation*
  • *Might also be able to leverage compliance testing suites*
ModBus Traffic Visualization
Future Work

• Example policy creation
  • And “policy builder”

• Enforcement actions
  • `iptables hooks`

• More advanced analytics processing
  • Smarter anomaly detection
  • Passive device profiling and determination
  • Network mapping

• Integration with Debian 8 Jessie x64
  • More testing needs to be done with `systemd` and 4.0 Linux Kernel before pushing to our repository.

• Bro/Broker
  • `Broccoli` is being phasing out and will be replaced by Broker.

• Visualization
  • Integration of web base monitoring with `bro-statsd` to aid in monitoring traffic of an ARMORE node.
Interested?

yardley@illinois.edu