Siemens Openlab Minor Review

March 2011

PLCs Security

An Automated Security & Robustness Testing Environment for Industrial Devices

Author: Filippo Tilaro

Supervised by: Brice Copy

Presentation Summary

- Industrial security model
  - IT VS ICS
  - Protocol Robustness
- Test bench components
  - PLC Monitoring to detect failures
  - Fuzzing Testing Implementation
- Creation of an automated testing system
  - Client-Server Model for the Fuzzing Framework

INDUSTRIAL SECURITY MODEL

IT vs ICS

Different **requirements**:

- Performances
  - best-effort vs real-time
- Availability
  - reboot strategy vs no downtimes admitted
- Network architecture
  - generic services (DNS, Domain Controller, …) vs industrial services
- Updating and patching
- Communication protocols
  - Public vs proprietary
- Software and component lifetime

Need for a specific Security Model

ICS failures **Consequences**:

- Loss of Vision (Temp or Perm)
- Loss of Control (Temp or Perm)

- In 2010 more than 180 security incidents were reported
  - Pipelines explosions
  - Environment damages
  - Casualties

We defined a specific security model for embedded devices based on ISA-99 security standards!

Security Model Fundamentals

- Access Control
- Data Integrity and Confidentiality
- Restricted Data Flow
- Auditing and logging
- Protocols Robustness
- Updating and Patching
- Backup and recovery
- Integration with third party systems and extensibility

Protocols Robustness Testing

IEEE defines robustness “*in the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions*.“

What is a robustness failure?

- Failure to return the expected packet
- Inability to progress to next protocol state
- Dropped connections
- Lost or modified data
- MORE IMPORTANT: Any unexpected effect in the process control!

Fuzzing and Grammar Testing

- Brute Force Testing:
  - Simple but inefficient
  - Not all the combinations are interesting

- Fuzzing:
  - Not random: need for debugging!
  - Not exhaustive but we can cover specific sequences
  - Grammar driven
  - Exploits the security specialists’ knowledge

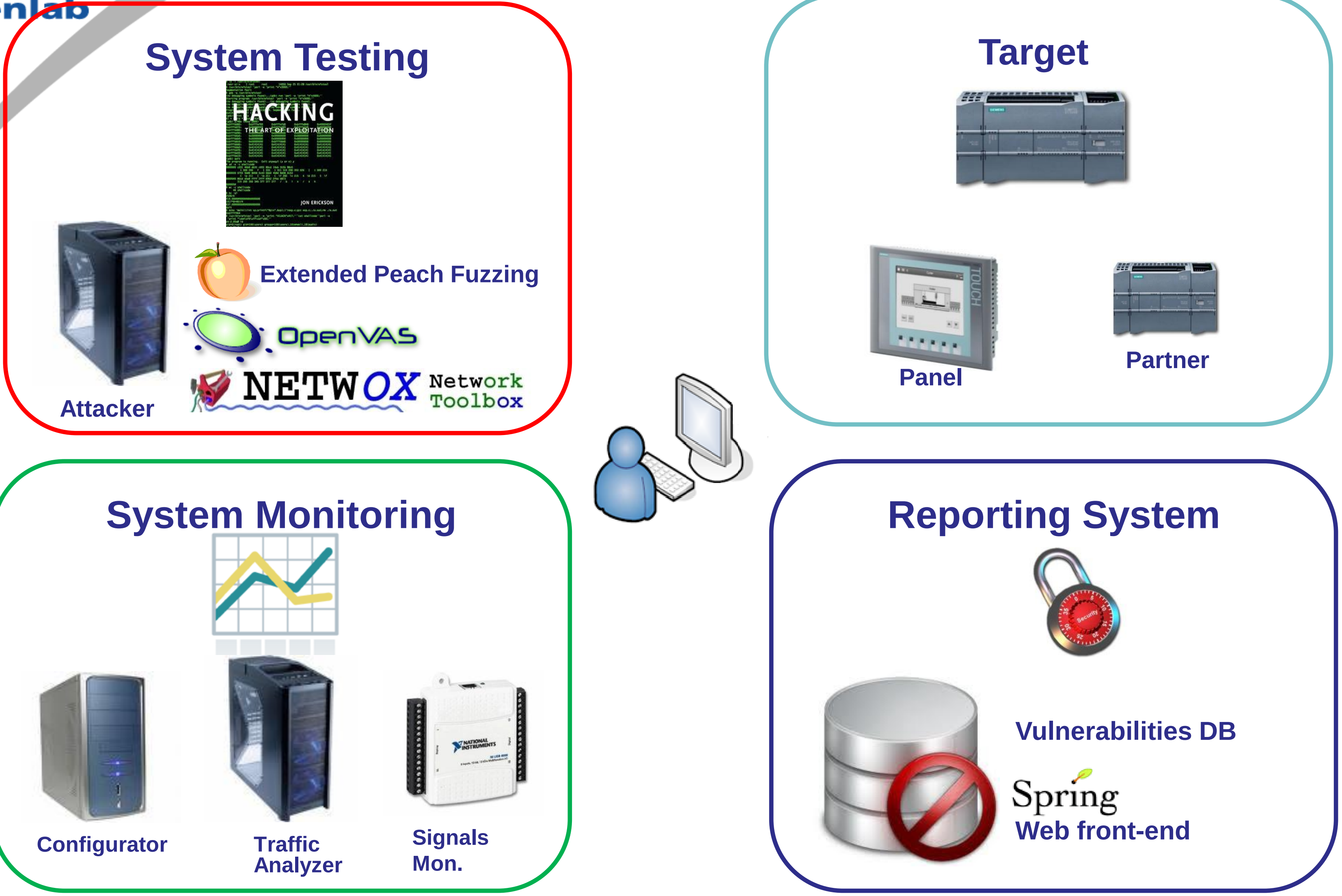
Protocol Testing Activities

- ISCI CRT-based 5 phases testing:
  1. Discover Protocol Functionalities and Attack Surface
  2. Storms and Maximum Load Tests
  3. Single Field Fuzzing
  4. Combinatorial Fields Injection
  5. Cross State Fuzzing (for Stateful Protocols)

We need a platform to automate all the testing activities!

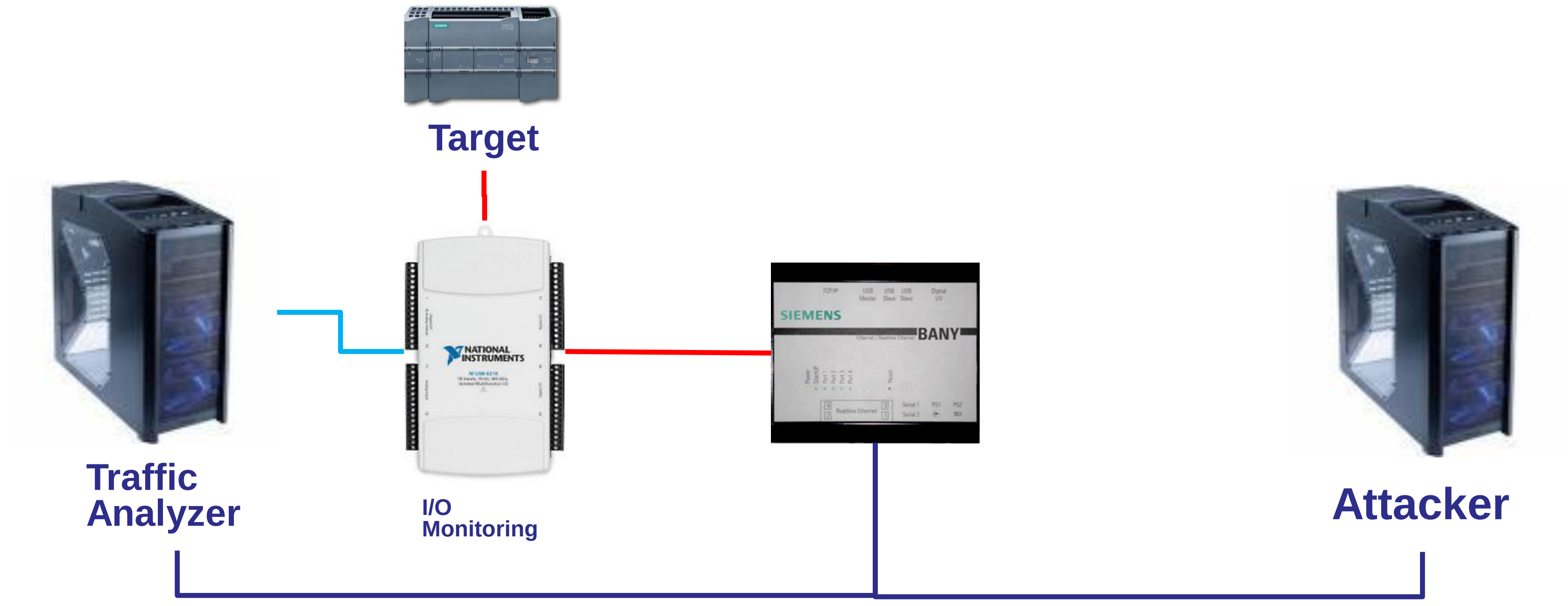
TEST BENCH COMPONENTS

Test-bench diagram



How do we detect a PLC failure?

- PLC’s I/O monitoring (through another PLC or DAC)
- Traffic Analysis
- PLC status



Fuzzing Testing requirements

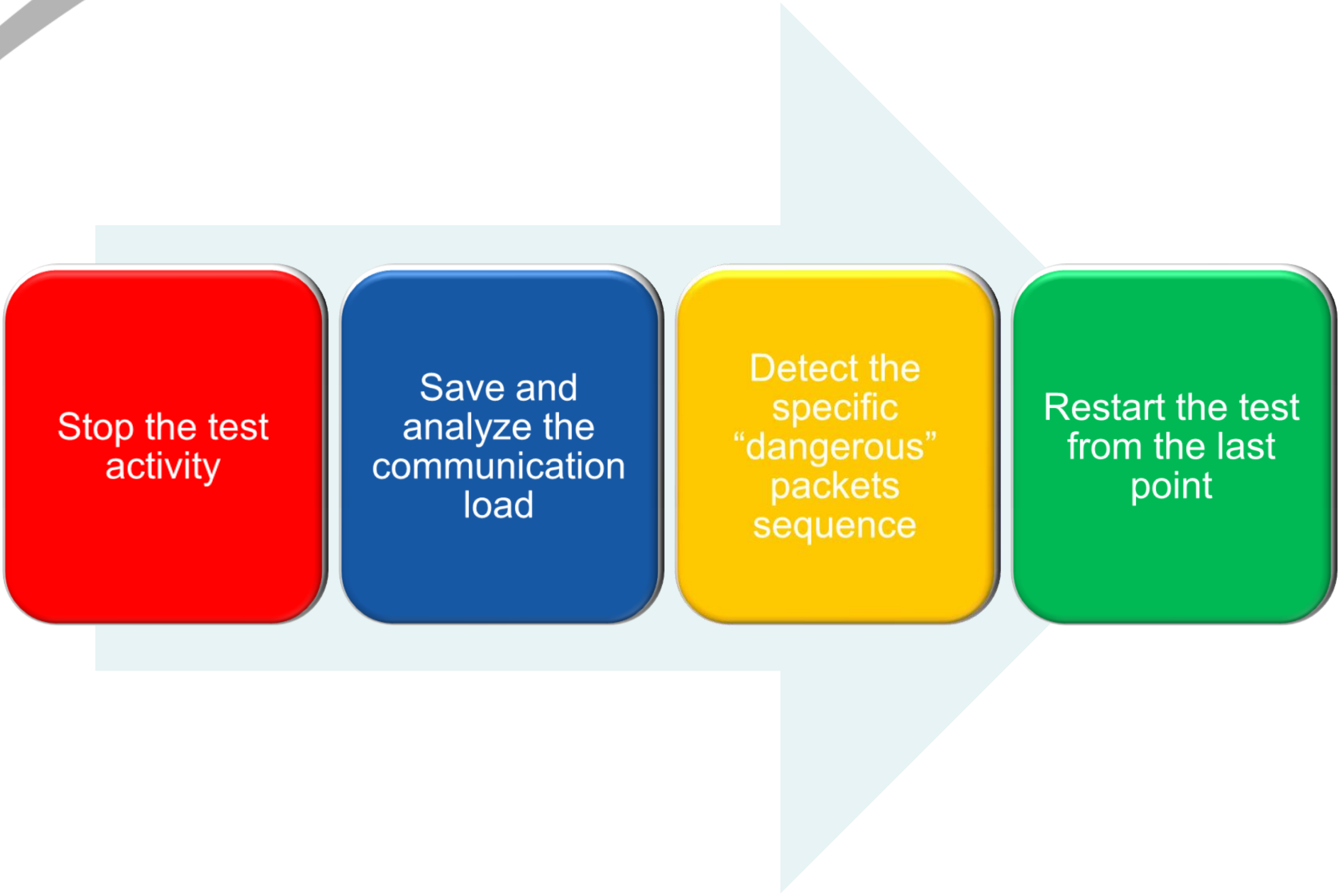
- A Common Framework:
  - No standalone scripts
- Scalability
  - Handle and organize the growing amount of tests (almost infinite combinations)
- Tests Customization
  - Protocol header format
  - Protocol field values
  - Protocol state machine
- Reproducibility
  - Essential for any debugging activity

Custom Fuzzing with Extended Peach

- Data Model:
  - Definition of the protocol header
  - Specify the protocol field values
  - Indicate protocol field to mutate or to calculate (checksum)
  - Specify field format (string or number) and codification (hex)
- Mutation Strategy
  - How to change the protocol fields values
- State Machine
  - In case of Stateful protocol
- Publisher
  - Send the specific protocol packet

CREATION OF AN AUTOMATED TESTING SYSTEM

How to react in case of PLC's failures?


Stop the test activity
Save and analyze the communication load
Detect the specific “dangerous” packets sequence
Restart the test from the last point

Test Environment Requirements

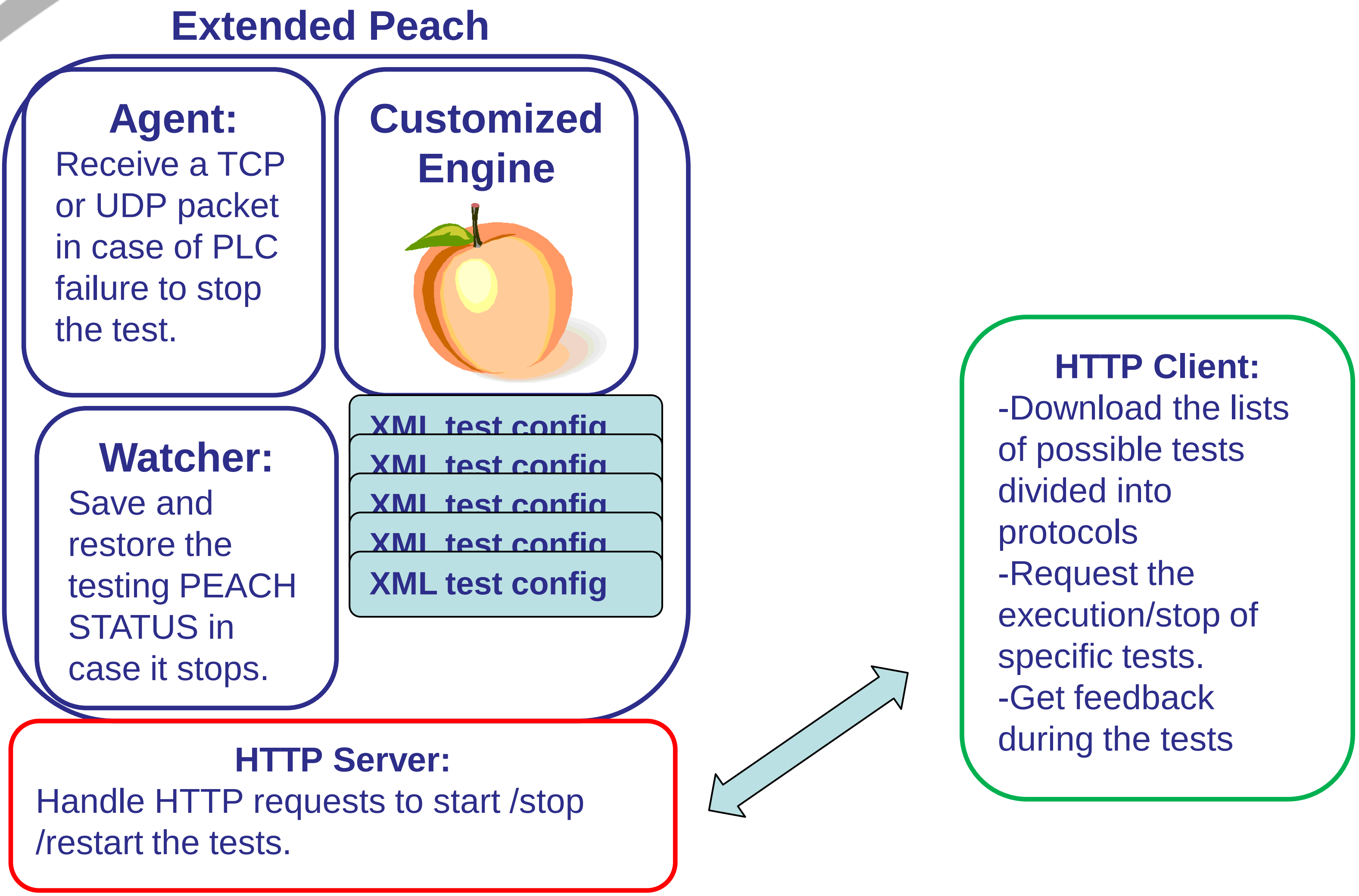
Test configuration:

- The security testers will not be able to change the tests but only to run it against specific targets
- The tests are built-in or produced as a part of the peach framework
- No specific security knowledge is necessary to run a test

Tests execution:

- No client-side installation required
- Client Compatibility with both Windows and Linux
- Automated Start/Stop of tests

Fuzzing Framework Architecture



Extended Peach

Agent:
Receive a TCP or UDP packet in case of PLC failure to stop the test.

Customized Engine

Watcher:
Save and restore the testing PEACH STATUS in case it stops.

HTTP Server:
Handle HTTP requests to start /stop /restart the tests.

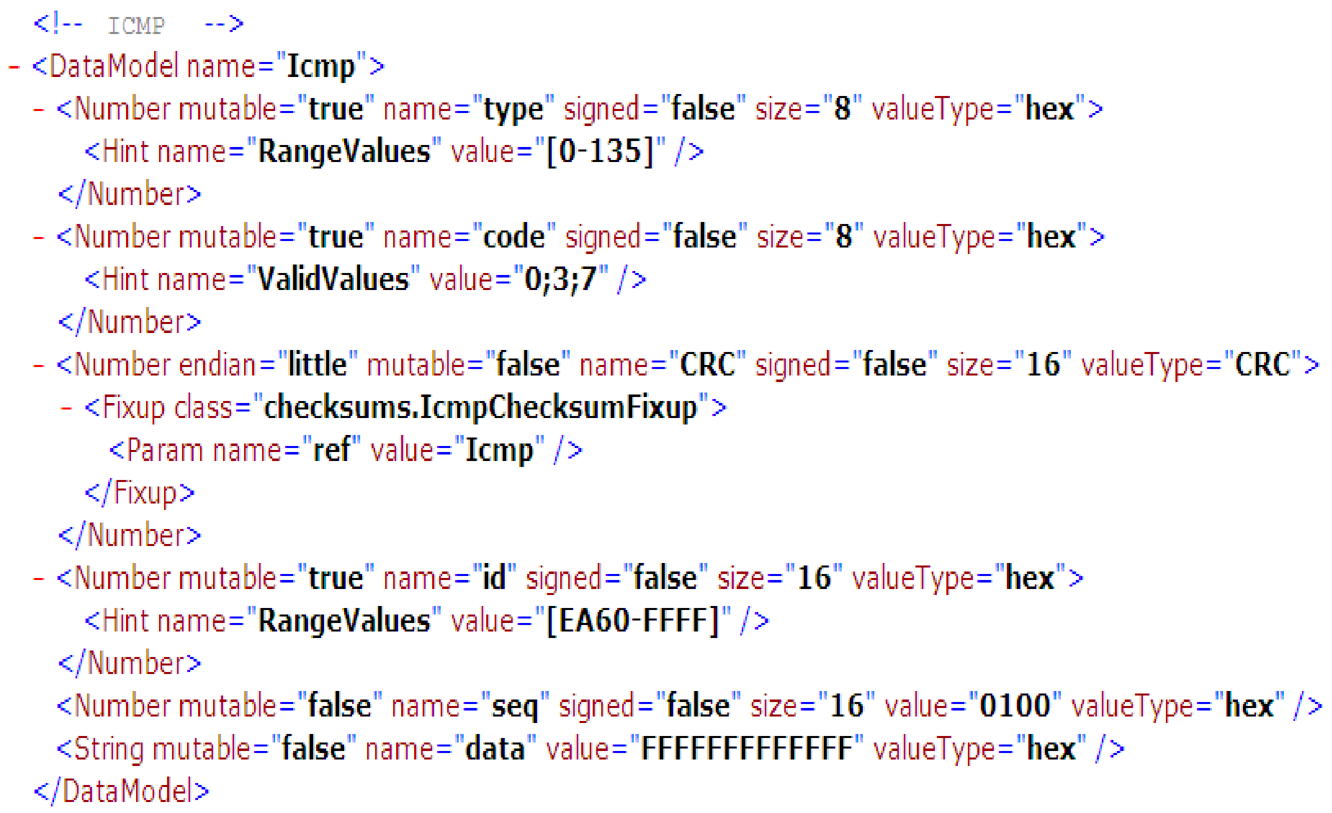
HTTP Client:
-Download the lists of possible tests divided into protocols
-Request the execution/stop of specific tests.
-Get feedback during the tests

Any Questions

Thank you for attending!

Custom Fuzzing with Extended Peach

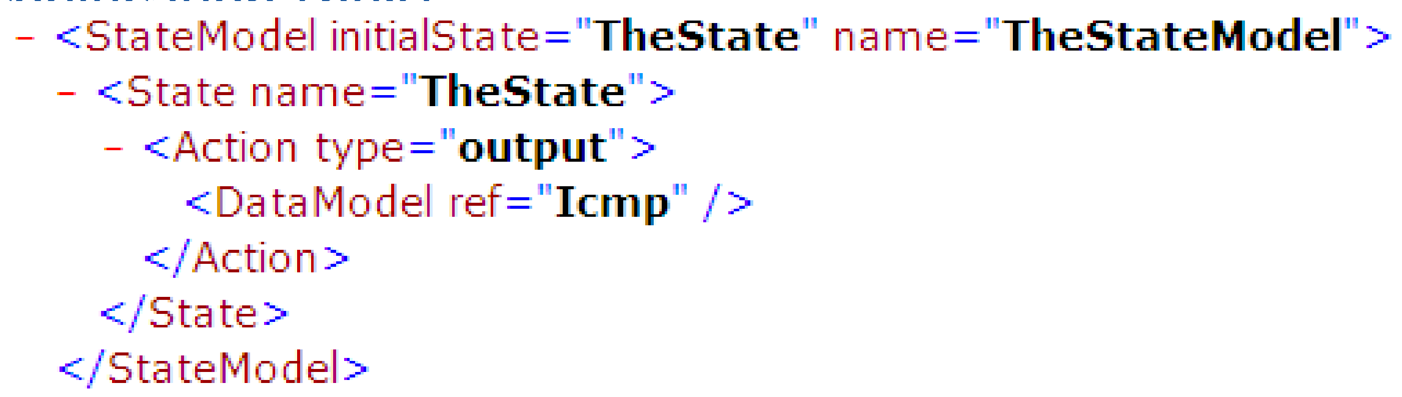
- Da
- Mu
- Sta
  - In case of Stateful protocol
- Publisher
  - Send the specific protocol packet



```
  <!-- ICMP -->
- <DataModel name="Icmp">
  - <Number mutable="true" name="type" signed="false" size="8" valueType="hex">
      <Hint name="RangeValues" value="[0-135]" />
    </Number>
  - <Number mutable="true" name="code" signed="false" size="8" valueType="hex">
      <Hint name="ValidValues" value="0;3;7" />
    </Number>
  - <Number endian="little" mutable="false" name="CRC" signed="false" size="16" valueType="CRC">
    - <Fixup class="checksums.IcmpChecksumFixup">
        <Param name="ref" value="Icmp" />
      </Fixup>
    </Number>
  - <Number mutable="true" name="id" signed="false" size="16" valueType="hex">
      <Hint name="RangeValues" value="[EA60-FFFF]" />
    </Number>
    <Number mutable="false" name="seq" signed="false" size="16" value="0100" valueType="hex" />
    <String mutable="false" name="data" value="FFFFFFFFFFFFF" valueType="hex" />
  </DataModel>
```

Custom Fuzzing with Extended Peach

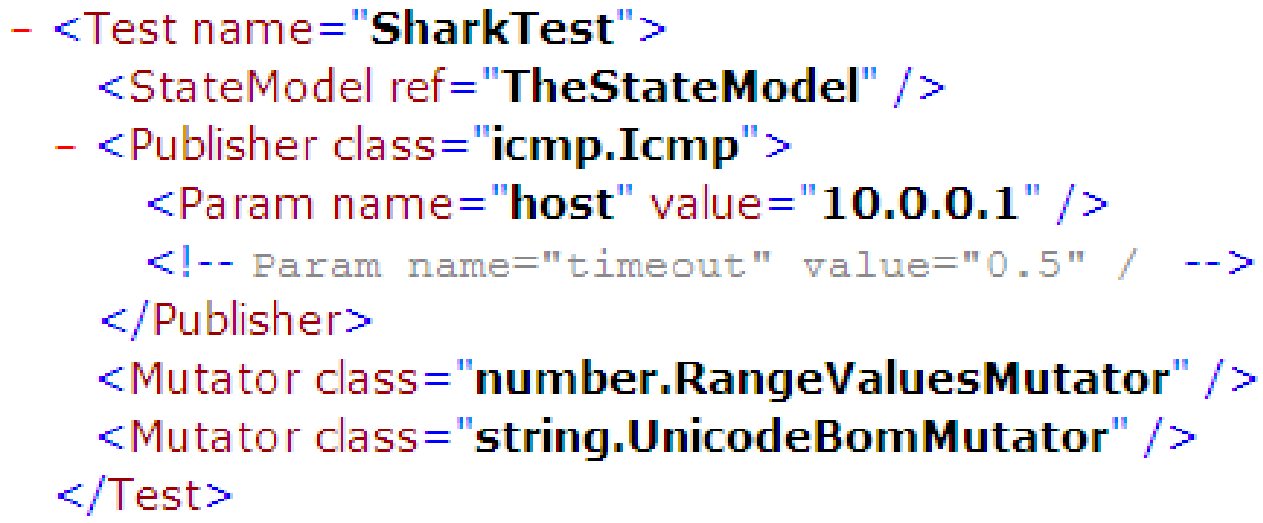
- Data Model:
  - Definition of the protocol header
  - Specify the protocol field values
  - Indicate protocol field to mutate or to calculate (checksum)
  - Specify field format (string or number) and codification (hex)
- Mu
  -
- Sta
  -
- Publisher
  - Send the specific protocol packet



```
- <StateModel initialState="TheState" name="TheStateModel">
  - <State name="TheState">
    - <Action type="output">
        <DataModel ref="Icmp" />
      </Action>
    </State>
  </StateModel>
```

Custom Fuzzing with Extended Peach

- Data Model:
  - Definition of the protocol header
  - Specify the protocol field values
  - Indicate protocol field to mutate or to calculate (checksum)
  - Spe
    coc
- Muta
  - Ho
- State
  - In
- Publisher
  - Send the specific protocol packet



```
- <Test name="SharkTest">
    <StateModel ref="TheStateModel" />
  - <Publisher class="icmp.Icmp">
      <Param name="host" value="10.0.0.1" />
      <!-- Param name="timeout" value="0.5" /  -->
    </Publisher>
    <Mutator class="number.RangeValuesMutator" />
    <Mutator class="string.UnicodeBomMutator" />
  </Test>
```