# ICS$^3$Fuzzer: A Framework for Discovering Protocol Implementation Bugs in ICS Supervisory Software by Fuzzing

Dongliang Fang, Zhanwei Song, Le Guan, Puzhuo Liu, Anni Peng, Kai Cheng, Yaowen Zheng, Peng Liu, Hongsong Zhu, Limin Sun

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

UNIVERSITY OF GEORGIA

中国科学院大学
University of Chinese Academy of Sciences

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE
30 1991-2021

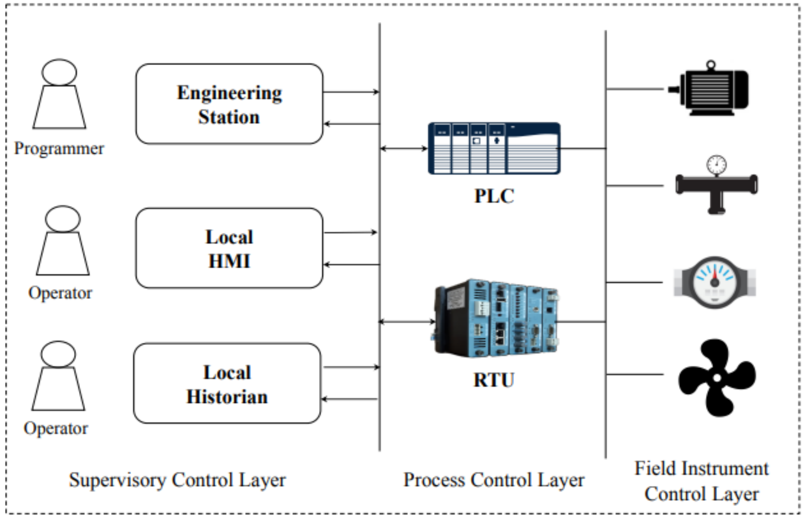
PennState
College of Information Sciences and Technology

# Outline

- Background
- Fuzzing Supervisory Software
- Design
- Evaluation
- Conclusion

# Background

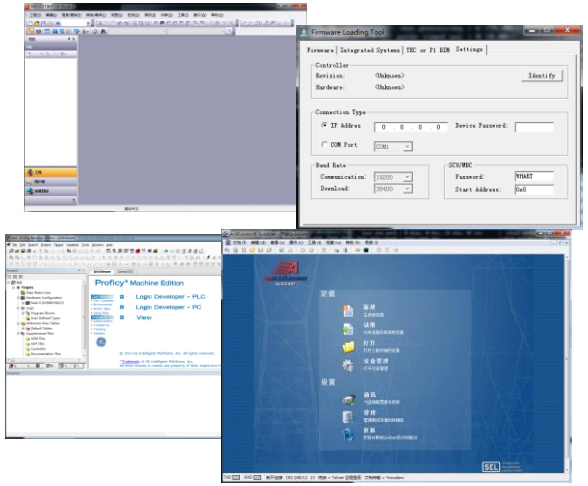
- **Industrial Control Systems (ICS)**
  - ICSs are widely used in critical infrastructures (e.g., chemical industry, power grid, nuclear plant, etc.)
  - Integration of IT and OT — multiple layers




- **ICS trends: isolated -> openness**
  - Standardized solutions
  - More connections -> larger attack surface
  - Facing the rapidly growing threats from cyber-attacks
    - ✓ Stuxnet (2010)
    - ✓ Ukraine Blackout (2015)
    - ✓ TSMC Ransomware (2018)
    - ✓ ……

# Background

- **Two targets in an ICS attack**
  - **Supervisory software**
    - Manage the physical devices, e.g., monitoring status, start/stop control, compile/download programs, etc.
    - E.g., management software such as Human Machine Interface (HMI), Engineering Software, and Configuration Software.



  - **Physical devices**
    - Directly interact with the physical world, e.g., accepting input from sensors, controlling on/off of switches, etc.
    - E.g., programmable Logic Controller (PLC), Remote Terminal Unit (RTU), and Programmable Automation Controller (PAC)

- **Existing research is geared towards embedded systems/devices; less has been studied on supervisory software**

# Background

- **Assumption – a powerful attacker who can …**
  - Access the inner network (e.g., via USB, insider attacker, etc.)
  - Monitor, intercept, and modify the network communication based on MITM attack
    - MITM attacks are commonly used in real-world ICS exploit, such as Stuxnet, IRONGATE.
- **Consequences when the supervisory software is compromised**
  - Present false data to the operator
  - Crash the supervisory software, making the control process out of monitor/control, real-time status cannot be updated
  - RCE by reusing existing control logic and existing authorizations
- **We focus on identifying the protocol implementation flaws within supervisory software**

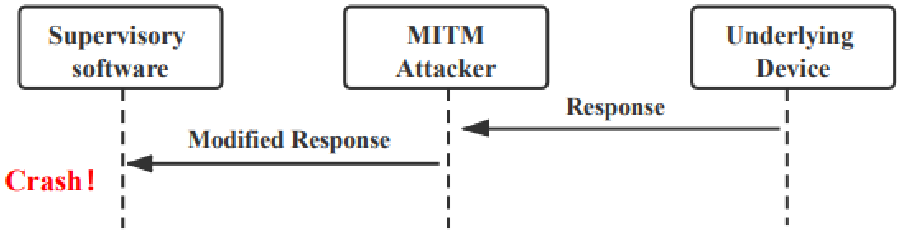



Figure 2: Crashing the supervisory software

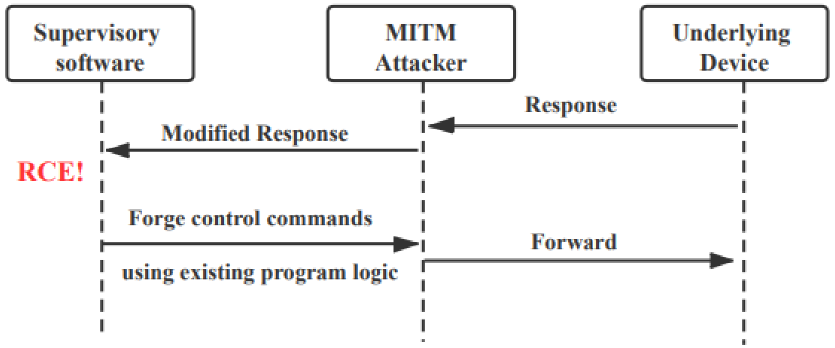



Figure 3: Forging malicious commands with RCE

# Fuzzing Supervisory Software

- **It is difficult to directly fuzz the commercial supervisory software**
  - Bulky size of executables (closed binary) running on the Windows System
  - GUI-driven
  - Client-role in the communication
  - Proprietary protocol involved

- **Existing solutions**

| Study | closed binary | client-role | GUI-management | Proprietary-protocols? |
|---|---|---|---|---|
| AFLNet [35] | ✗ | ✗ | ✗ | ✗ |
| PripFuzz [32] | ✓ | ✗ | ✗ | ✓ |
| Pulsar [15] | ✓ | ✓ | ✗ | ✓ |
| Polar [25] | ✗ | ✗ | ✗ | ✗ |
| Peach* [26] | ✗ | ✗ | ✗ | ✗ |
| Kif [1] | ✓ | ✗ | ✗ | ✓ |
| Achilles [41] | ✓ | ✗ | ✗ | ✗ |

# Fuzzing Supervisory Software

- **Major challenge: the tightly coupled of GUI and proprietary protocol implementation**

| Study | closed binary | client-role | GUI-management | Proprietary-protocols? |
|---|---|---|---|---|
| AFLNet [35] | ✗ | ✗ | ✗ | ✗ |
| PripFuzz [32] | ✓ | ✗ | ✗ | ✓ |
| Pulsar [15] | ✓ | ✓ | ✗ | ✓ |
| Polar [25] | ✗ | ✗ | ✗ | ✗ |
| Peach* [26] | ✗ | ✗ | ✗ | ✗ |
| Kif [1] | ✓ | ✗ | ✗ | ✓ |
| Achilles [41] | ✓ | ✗ | ✗ | ✗ |

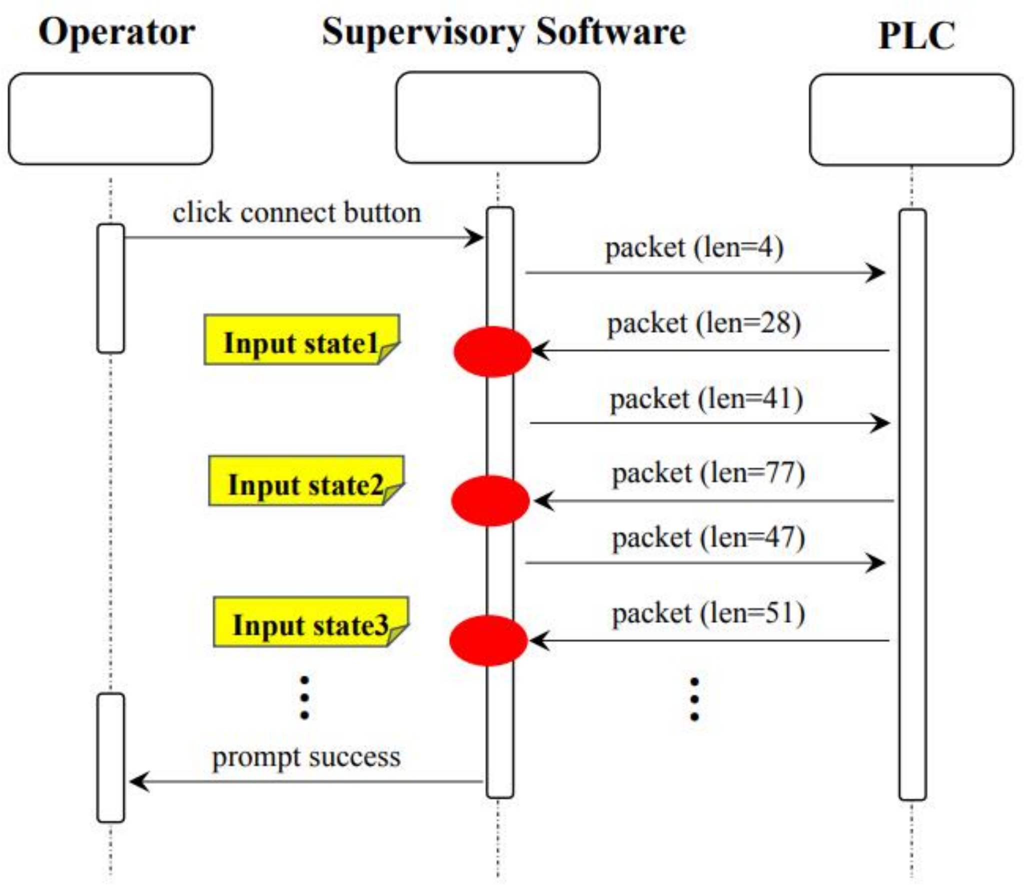
- **Writing a harness is extremely difficult**
  - Not general and requires many manual work
  - WINNIE supports writing harness within ONLY two components
  - Difficult to fuzz deep state-space of a proprietary protocol

- **Our approach:** we run and fuzz **the whole supervisory software**, with the synchronized controls of GUI operations and network communication

# Idea

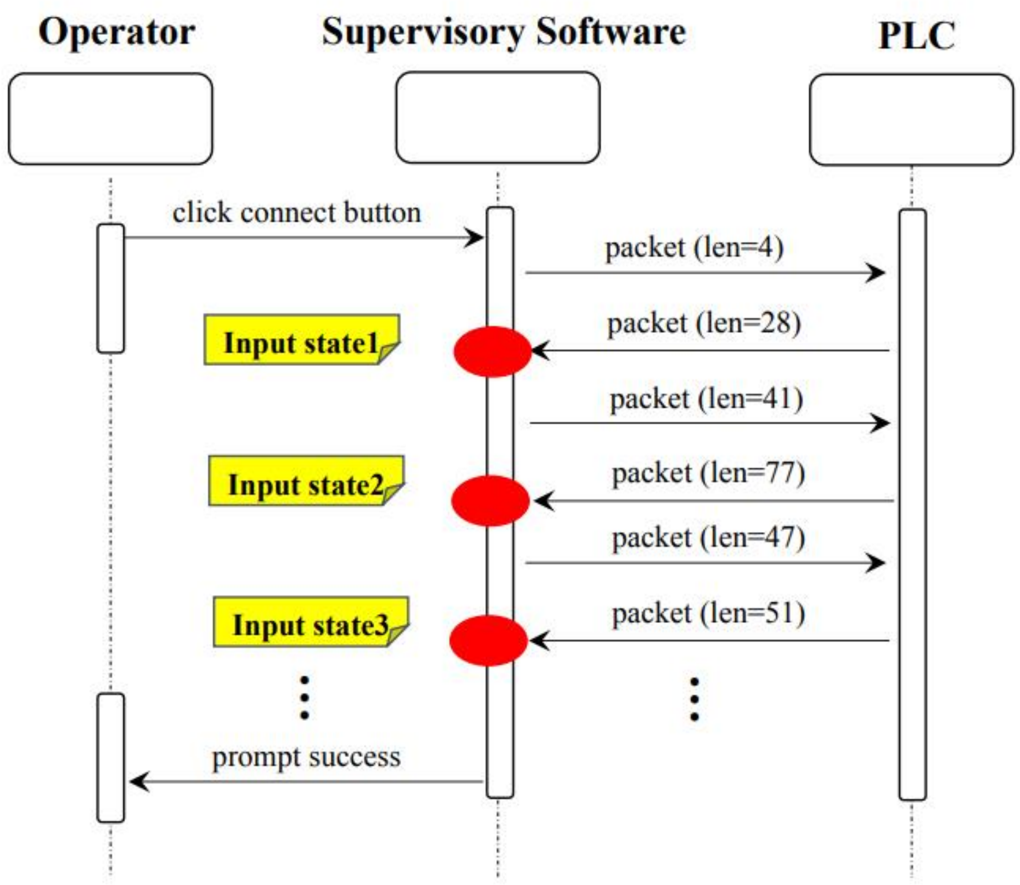
- **Example: protocol states involved in a function**
  - **Start:** click the button, and then the packet are exchanged in both directions
  - **Middle:** additional button-pushing operations
  - **End:** Periodically exchanging heartbeat message or stopping the message exchanges.

- **Input state≈protocol state**
  - An input state is determined by the previous button-pushing operations and the previous input states.



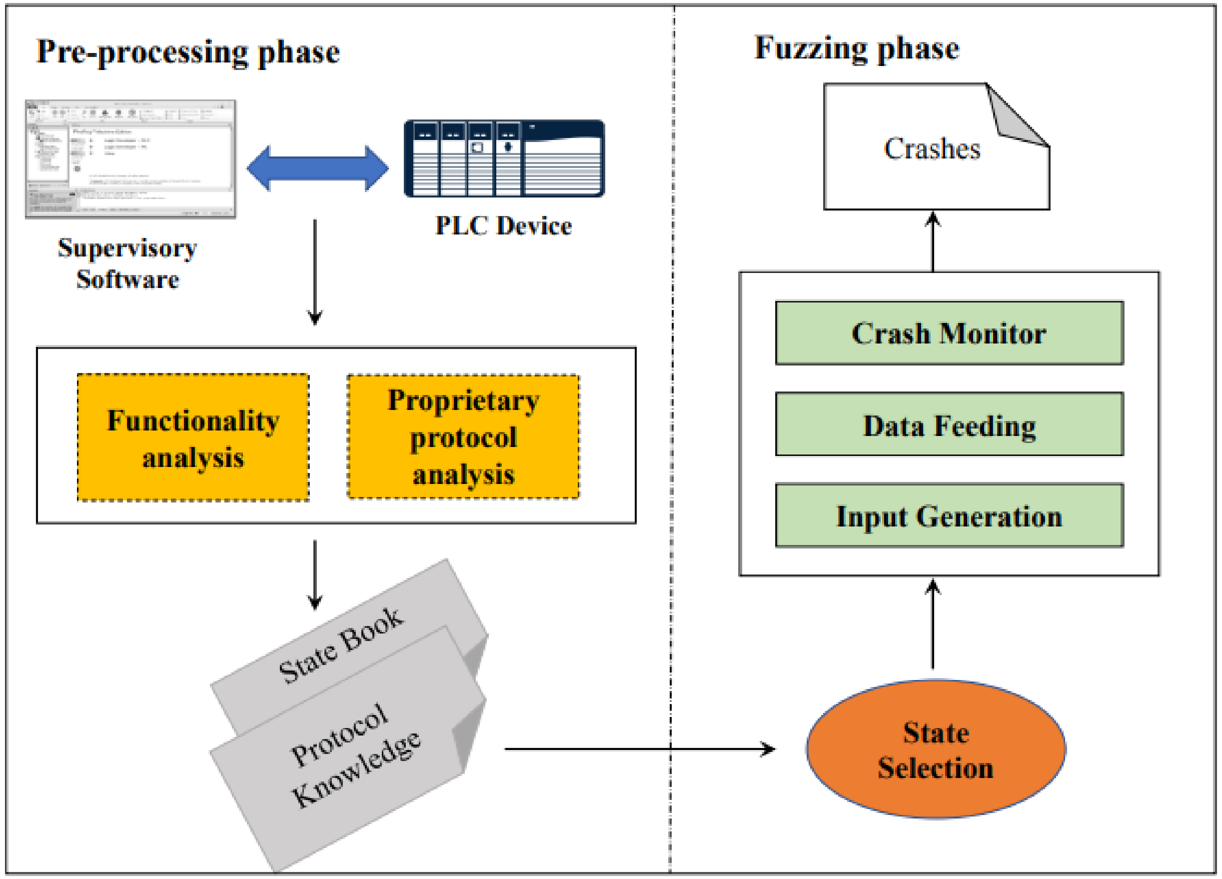
# Design Challenges

- **C1: How to enter a specific input state**
  - Each input state is reachable and testable
- **C2: Unknown message frame format and state-space in proprietary protocols**
  - Producing effective inputs and fuzzing valuable input states with higher priority
- **C3: Real device involved**
  - Simulate the session of proprietary protocol to making the fuzzing process scalable



# Design Overview

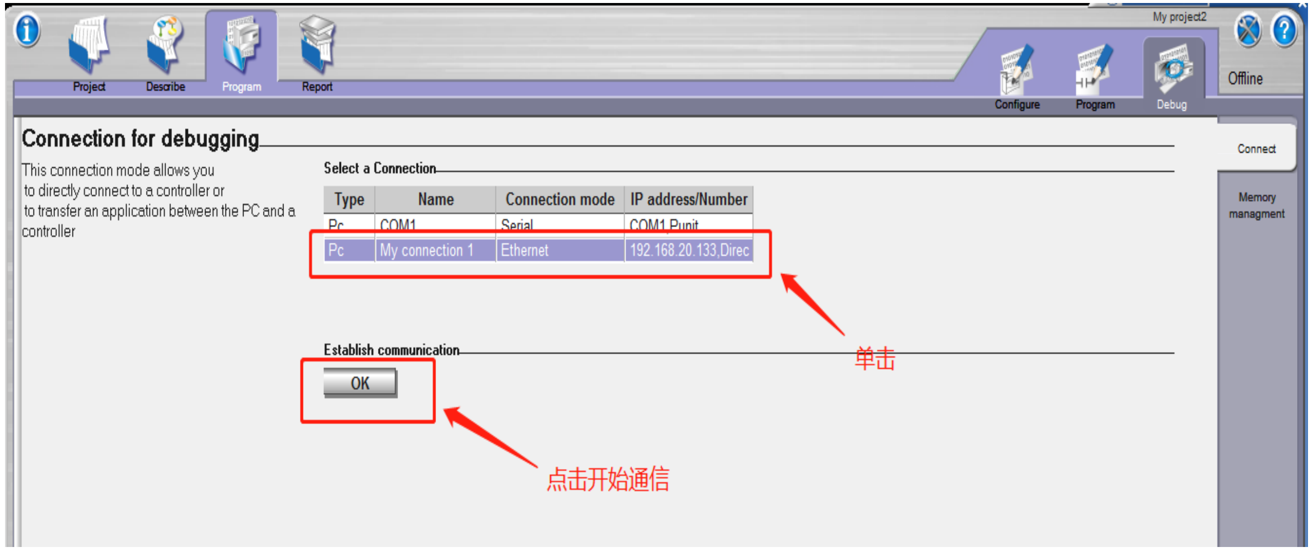
- **Two phases**
  - Pre-processing phase
    - ✓ functionality analysis
    - ✓ Proprietary protocol analysis
  - Fuzzing phase — Fully automated
    - ✓ State selection
    - ✓ Input generation
    - ✓ Data Feeding
    - ✓ Crash monitor
- **Design choice: blackbox vs greybox**
  - We support both
  - However, instrumenting bulky GUI-driven executables incurs huge overhead (~30X)



# Functionality analysis

- **Identify UI triggers of functionality**
  - Find GUI operations that lead to the network events
  - Capture the corresponding messages

- **Prepare the GUI operation triggers**
  - Record the GUI operation orders
  - Preparing the UI elements triggers (guiautolits)




```
Local $choose_windw = WinWaitActive("TwidoSuite","", 1)
If $choose_windw Then
    ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button212")
EndIf
Sleep(800)
ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button212")
ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button16")

ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button22")
; choose project
Local $bPos = WinGetPos("TwidoSuit", "")
Local $bx = $bPos[0] + 68 + 33
Local $by = $bPos[1] + 415 + 50
MouseClick($MOUSE_CLICK_LEFT, $bx, $by, 1, 1)
ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button25")
Sleep(300)
ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button680")
ControlClick("[TITLE:TwidoSuite;CLASS:#32770]","CAS_Button", "Button692")
```

# Proprietary Protocol analysis

- **Infer Protocol Formats –** generate effective inputs
  - Leveraging an existing tool Netzob
- **Obtain State-space –** switch more valuable input states
  - Identify and distinguish input states
  - Filter repeated behaviors (e.g., the input states related to the heartbeat messages )
  - Each input state is distinguished/measured by
    - ✓ origin message
    - ✓ input index
    - ✓ the corresponding execution trace (based on DynamoRIO framework)
- **Device Emulation –** make fuzzing more scalable
  - The Fuzzer needs to act as a PLC device role to feed the test cases
  - For each request from the supervisory software, simulating a response needs to
    - ✓ Identify the corresponding response message in the captured traffic
    - ✓ Adjust the dynamic field such as session ID, sequence number, etc

# State Selection

- **Select the most promising input state to fuzz against**
  - “deeper” network communication
  - More basic block executed under the state
  - More complex the input

- **Calculate a weight for each state**
  - depth: message index in an interaction
  - bb_cnt: execution trace when handling a packet
  - fld_cnt: the count of reverse-engineered field in a message (to represent the complexity of the input)

$$w_i = \frac{1}{3}\left(\frac{depth_i}{\sum_{j=0}^{N-1} depth_j} + \frac{bb\_cnt_i}{\sum_{j=0}^{N-1} bb\_cnt_j} + \frac{fld\_cnt\_i}{\sum_{j=0}^{N-1} fld\_cnt\_j}\right)$$

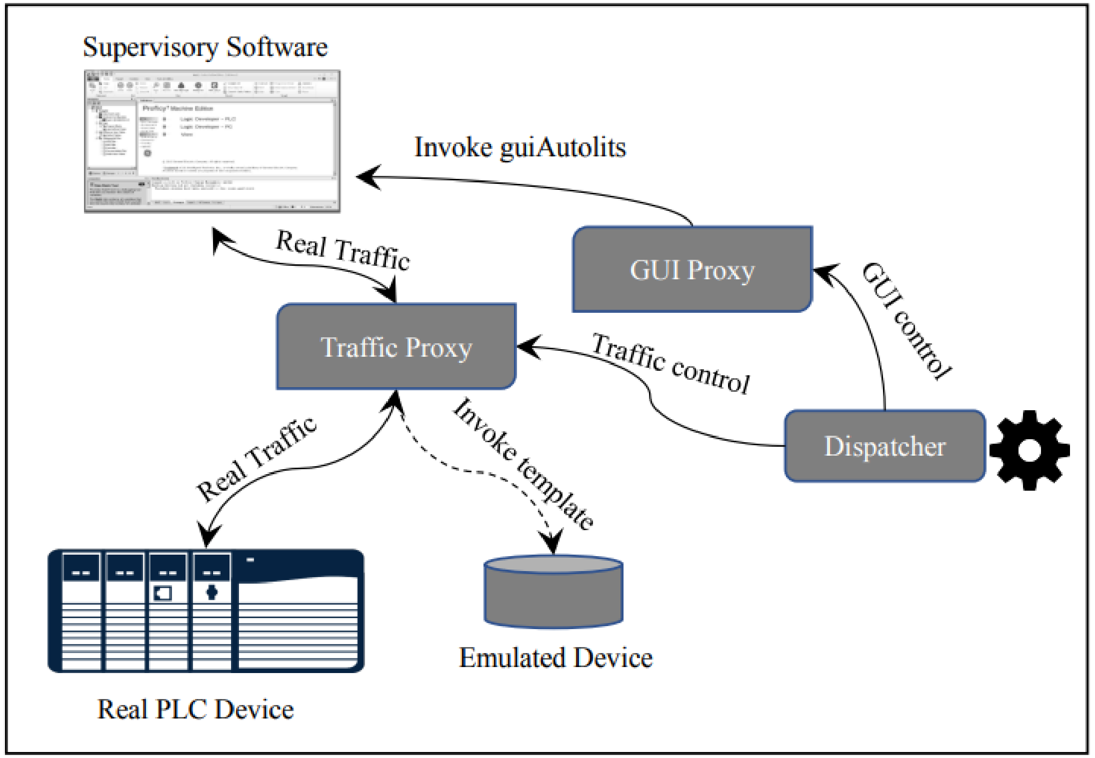
**Algorithm 1** State-selection based fuzzing loop

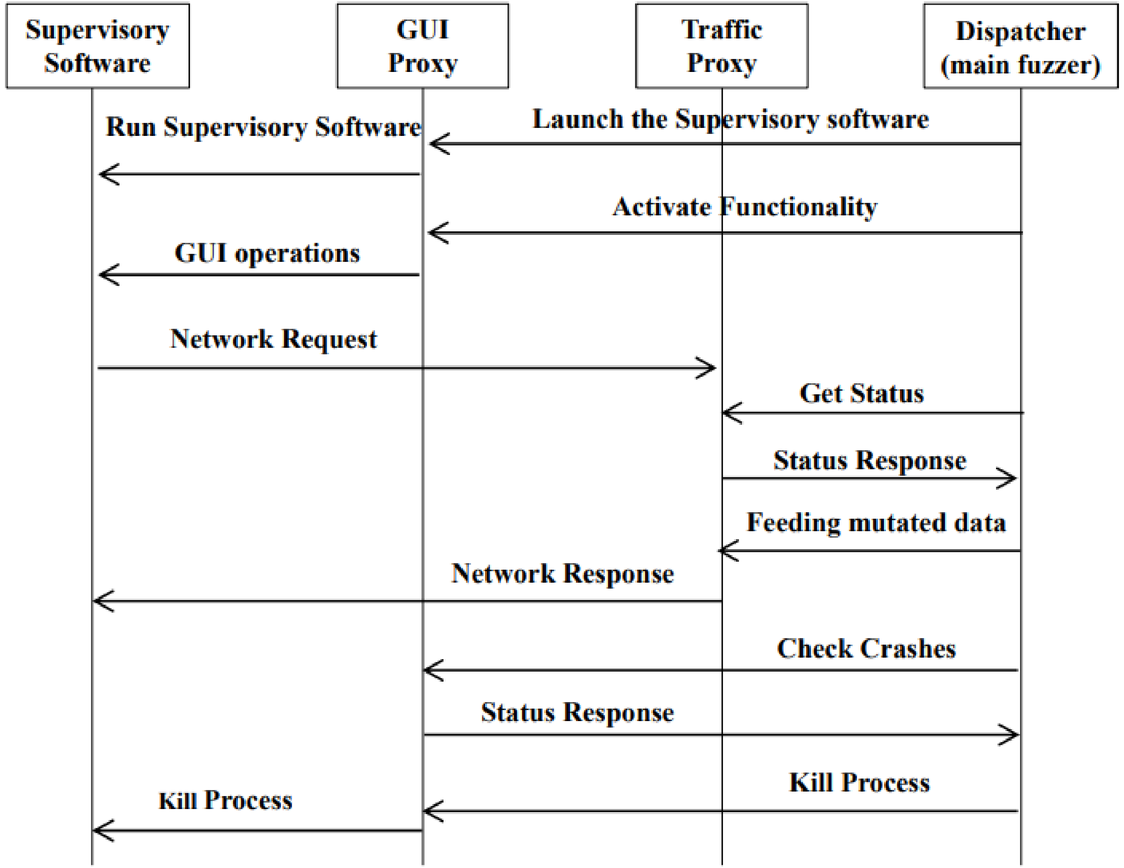
**Input:** Limited state count for a functionality, $N$
Limited test case for a specific input state, $M$
State book of a functionality, $S$
**Output:** Crashes Record, $C$

```
states ← FILTERED(S)
weights ← GETWEIGHTS(states)
for i ← 0; i < N; i++ do
  cur_state ← CHOOSE(states, weights)
  j ← 0
  while j < M do
    mutated_input ← GENERATE(cur_state)
    result ← RUN(cur_state, mutated_input)
    if IsCrash(result) then
      item ← TUPLE(cur_state, mutated_input)
      C ← APPEND(C, item)
    end if
    j ← j + 1
  end while
end for
return C
```

# Input Feeding

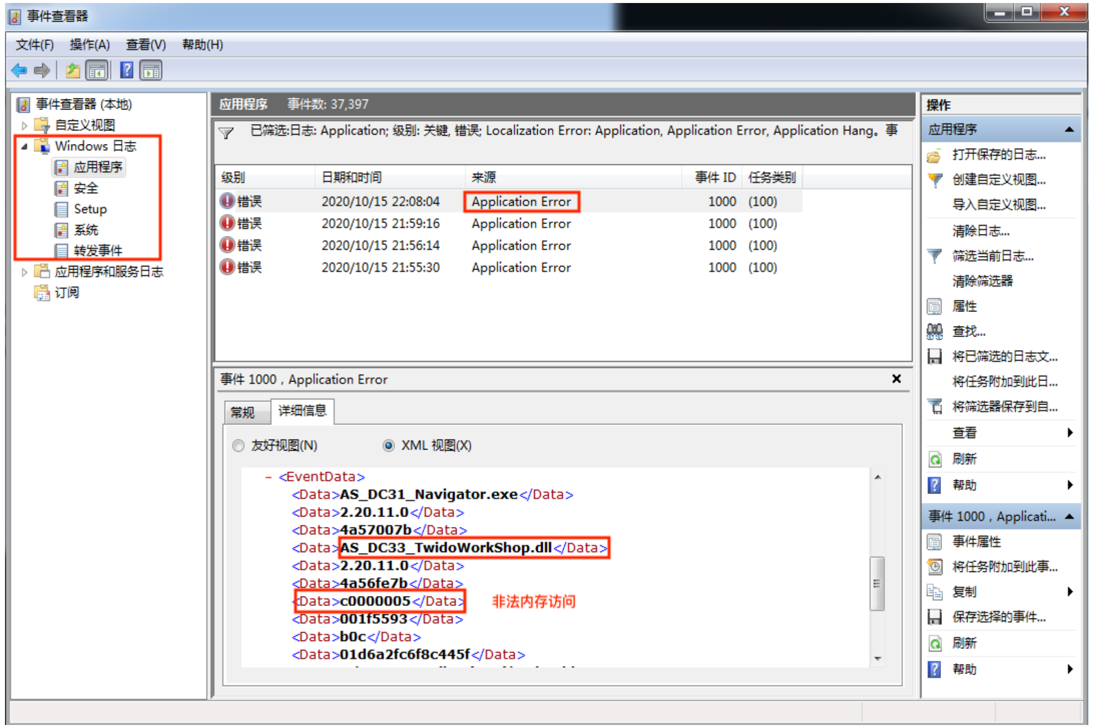
- **Data Feeding**
  - Two proxies: GUI proxy and Traffic proxy
  - A dispatcher commands the two proxies



# Crash Monitor

- **Check the Eventlog after feeding the testcase**
  - We do not consider liveness of the connection as an indicator, because the normal communication could close the connection
  - Once an application crashes, a record will be added in the Eventlog of Windows System with a tag "*Application Error*"




- **Program hangs**
  - They are not considered as a bug, because whether a long delay should be considered as a bug or not depends on specific scenarios
  - There is no standard to specify the cutoff value to separate a normal response and a delayed response
  - Record the hangs can be easily enabled in the current prototype

# Evaluation - Effectiveness

- **Four targets**

| Vendor | Software | Version | Image Size | Device | Proprietary Protocol? (Yes,No) |
|---|---|---|---|---|---|
| Mitsubishi | GX Works2 | 1.591 | 671M | Q06UDEHCPU | Yes |
| Emerson | Proficy Machine Edition | 9.0 | 1200M | GE RX 7i | Yes |
| Schneider | TwidoSuite | 2.20.11 | 79M | TWDLCAE40DRF | Yes |
| Panasonic | FPWIN GR | 2.95 | 106M | FP-X C60R | Yes |

- **Bug identified**
  - 13 0-day bugs and got 3 CVEs
  - Among them, 2 bugs are "CRITICAL" (high risk, basic score is 9.8) and 40 different products are affected

| Software | Vulnerability Type | Number | status |
|---|---|---|---|
| GX Works2 | HeapOverflow<br>Unknown crash | 6<br>1 | confirmed |
| Proficy Machine Edition | NullPointer Dereference<br>Unknown crash | 1<br>3 | confirmed |
| TwidoSuit | Unknown crash | 1 | confirmed |
| FPWIN GR | Unknown crash | 1 | confirmed |

# Evaluation - Effectiveness

- **Effectiveness of pruning input states**

| Software | # Selected session | # Origin states | # Post-pruned states |
|---|---|---|---|
| GX Works2 | 8 | 326 | 93 |
| Proficy Machine Edition | 8 | 842 | 52 |
| TwidoSuit | 2 | 136 | 27 |
| FPWIN GR | 4 | 200 | 31 |
| TOTAL | 22 | 1504 | 203 |

- **Effectiveness of state selection**
  - There is no fuzzer designed for testing supervisory software
  - We reused the same fuzzing framework and use a random strategy to select a input state during fuzzing
  - ICS3Fuzzer can find 5 more bugs

# Evaluation - Performance

- **Time-cost breakdown (mainly composed of four parts)**
  - Restarting the supervisory software
  - Operating the GUI
  - Network communication
  - The others

| Software | Launch | GUI Operations | Network Communication | others |
|---|---|---|---|---|
| GX Works2 | 1.095 | 2.363 | 1.850 | 3.415 |
| Proficy Machine Edition | 4.115 | 4.349 | 9.374 | 5.356 |
| FPWIN GR | 1.114 | 0.361 | 1.274 | 2.371 |
| TwidoSuite | 6.765 | 0.592 | 5.444 | 2.157 |

- **Why not use feedback-based method?**
  - It needs 34.9s to restart the instrumented GX Works2 and the number is 1.1s without instrumentation
  - 48hours can only generate 4.5K testcases, it is hard for the genetic algorithm to make progress
  - Most of the test cases (more than 95%) cannot make through the initial input state

- **Why not use snapshot-based method?**
  - Snapshot/rollback of a complete Windows system is very slow, and it almost costs 32 seconds for a test case
  - The connection is easy to be closed due to slow network recovery, and it is hard to feed the testcase

# Conclusions

- **Contribution**
  - We designed and implemented ICS3Fuzzer, which is an insecurity testing framework specific for the supervisory software in ICS
  - We propose a new fuzzing strategy, which selects input states based on execution trace and corresponding inputs
  - Our tool found 13 real world bugs and received 3 CVEs, 2 of them are classified as critical, 40 different products are affected

- **Feature work**
  - Feedback-based method can be improved in terms of high fuzzing speed (e.g., via parallelization)
  - Better protocol state management mechanisms can be proposed

# Thank you!

# Q&A

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS