Trustworthy Critical Infrastructures

Threats, Challenges, and Countermeasures

Saman Zonouz

IFIP WG 10.4 - Jan. 2016

Outline

- Background
  - *Cyber-physical critical infrastructures*
  - *Potential threats*
- Our Solutions
  - *Trustworthy sensing*
  - *Adaptive intrusion tolerance*
  - *Control command verification*
- Other contributions
- Conclusions and Q&A

The New York Times

WORLD U.S. N.Y. / REGION BUS

AFRICA AMERICAS ASIA PACIFIC

Panetta Warns of D

By ELISABETH BUMILLER and THOM SHAN

Published: October 11, 2012 | 192 Comme

Defense Secretary Leon E. Par
United States was facing the p
and was increasingly vulnerab
could dismantle the nation's p
financial networks and govern

Double thre
two fronts

CNBC

W FP

Peace | Energy | Iraq | NSA ABOUT

EDITION: INTERNATIONAL U.S. MÉXICO ARABIC

TV: CNNi CNN en Español

Set edition preference

CNN

Home

Hon
cyb

By Suzan

July 4, 201

The In

Money

my Tech Personal Finance

lobile | Security | Social | In

CNNMoney

more

Nextgov

TRENDING ▸ HEALTHCARE.GOV // BROKEN WARRIORS // OPEN DATA /

INDUSTRY NEEDS TO STEP UP TO PROTECT THE POWER GRID FROM CYBER ATTACK

e Tra
rica S

em

EUROPICS

of
f a school
und and

ICS-CERT MONITOR

October/November/December 2012

U.S. DEPARTMENT OF HOMELAND SECURITY

INDUSTRIAL CONTROL SYSTEMS CYBER EMERGENCY RESPONSE TEAM

CONTENTS

INCIDENT RESPONSE ACTIVITY

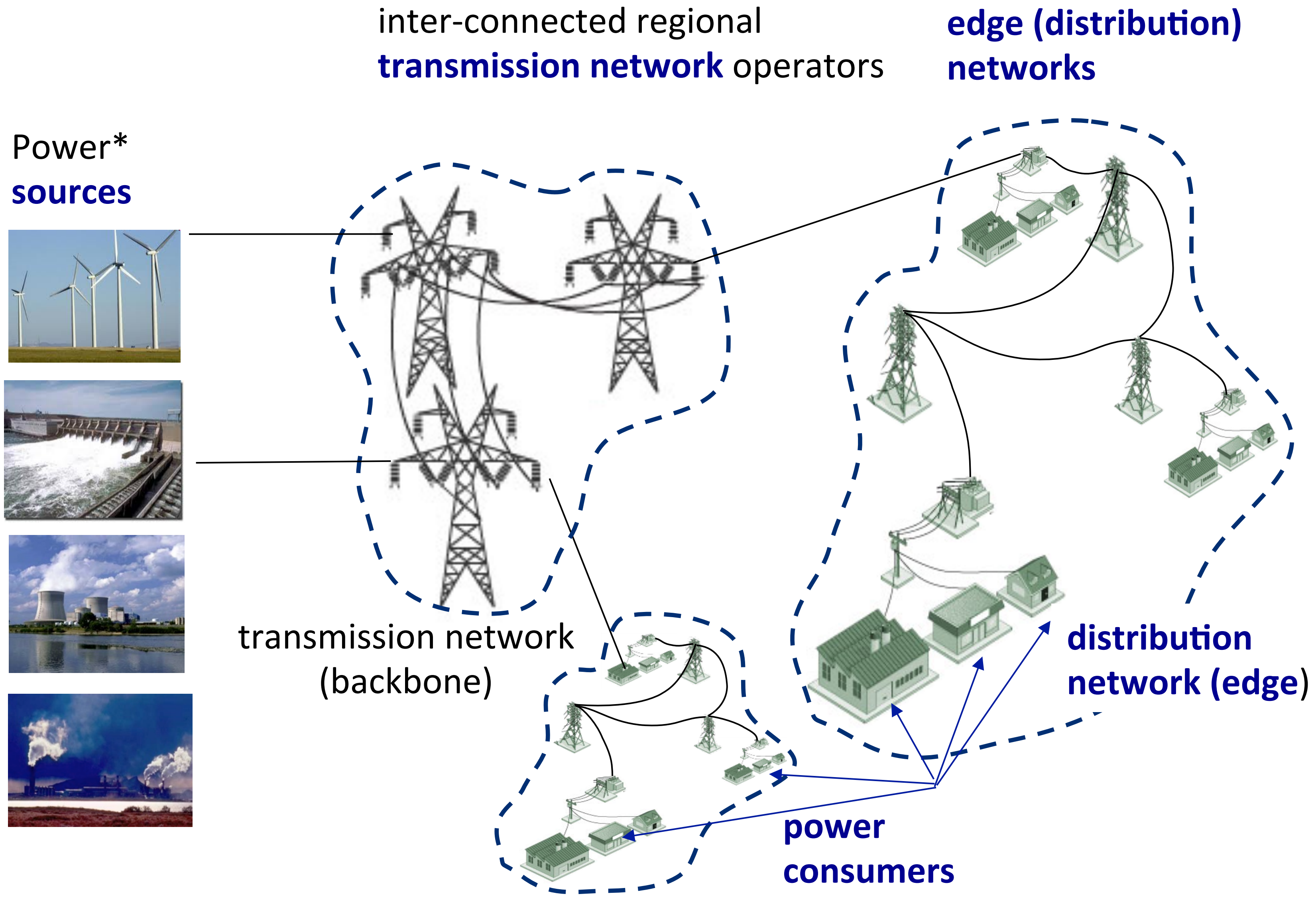
MALWARE INFECTIONS IN THE CONTROL ENVIRONMENT

ICS-CERT recently provided onsite support at a power generation facility where both common and sophisticated malware had been discovered in the industrial control system environment. The malware was discovered when an employee asked company IT staff to inspect his USB drive after experiencing intermittent issues with the drive's operation. The employee routinely used this USB drive for backing up control systems configurations within the control environment.

When the IT employee inserted the drive into a computer with up-to-date antivirus software, the antivirus software produced three positive hits. Initial analysis caused particular concern when one sample was linked to known sophisticated malware. Following analysis and at the request of the customer, an onsite team was deployed to their facility where the infection occurred.

ICS-CERT's onsite discussions with company personnel revealed a handful of machines

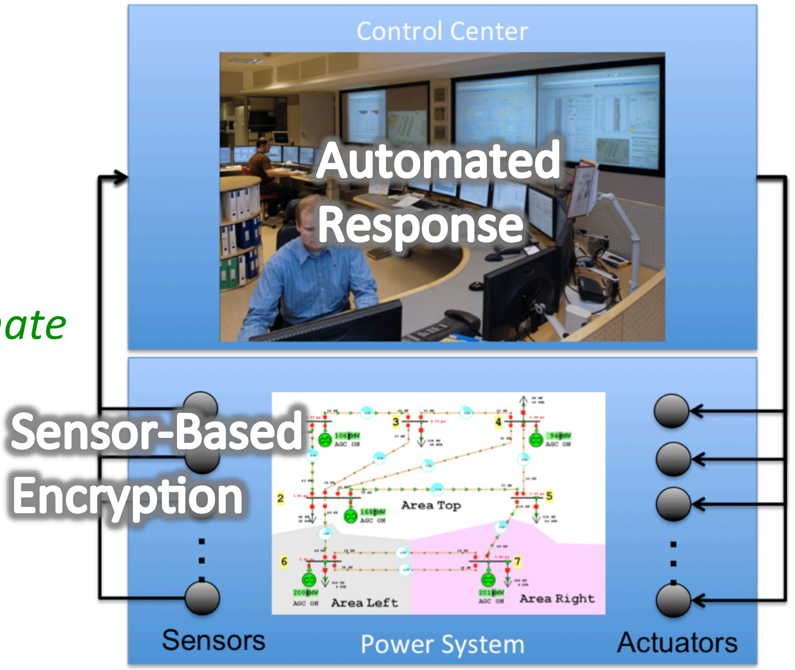
The Electric Grid Structure


inter-connected regional transmission network operators
edge (distribution) networks
Power* sources
transmission network (backbone)
distribution network (edge)
power consumers

*Jim Kurose, Networking Challenges for the Smart Grid, IIT Mumbai, 2013.

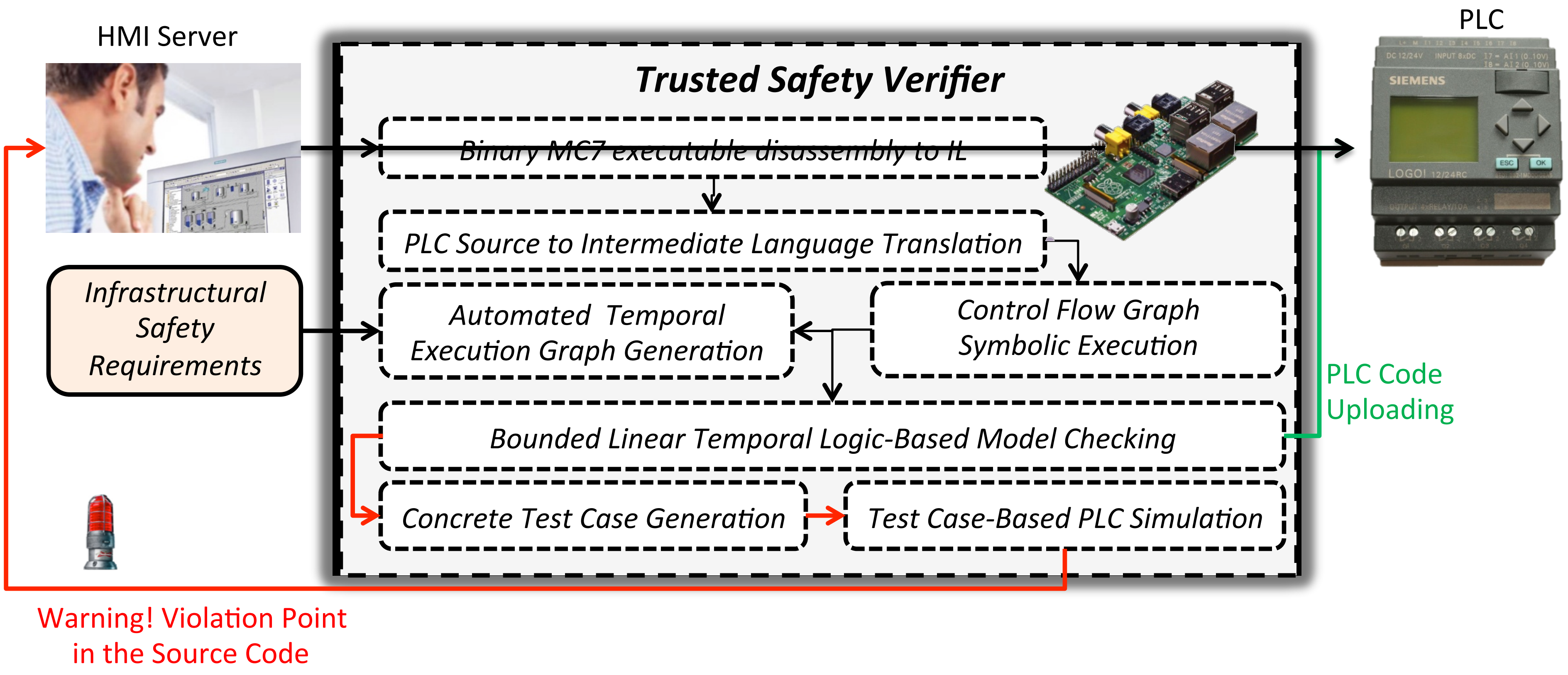
Control Command Verification

- Problem
  - *Malicious control command injection*
- Solution
  - *Verify if the control command is legitimate*
- Challenge
  - *Not having to check every single input?*

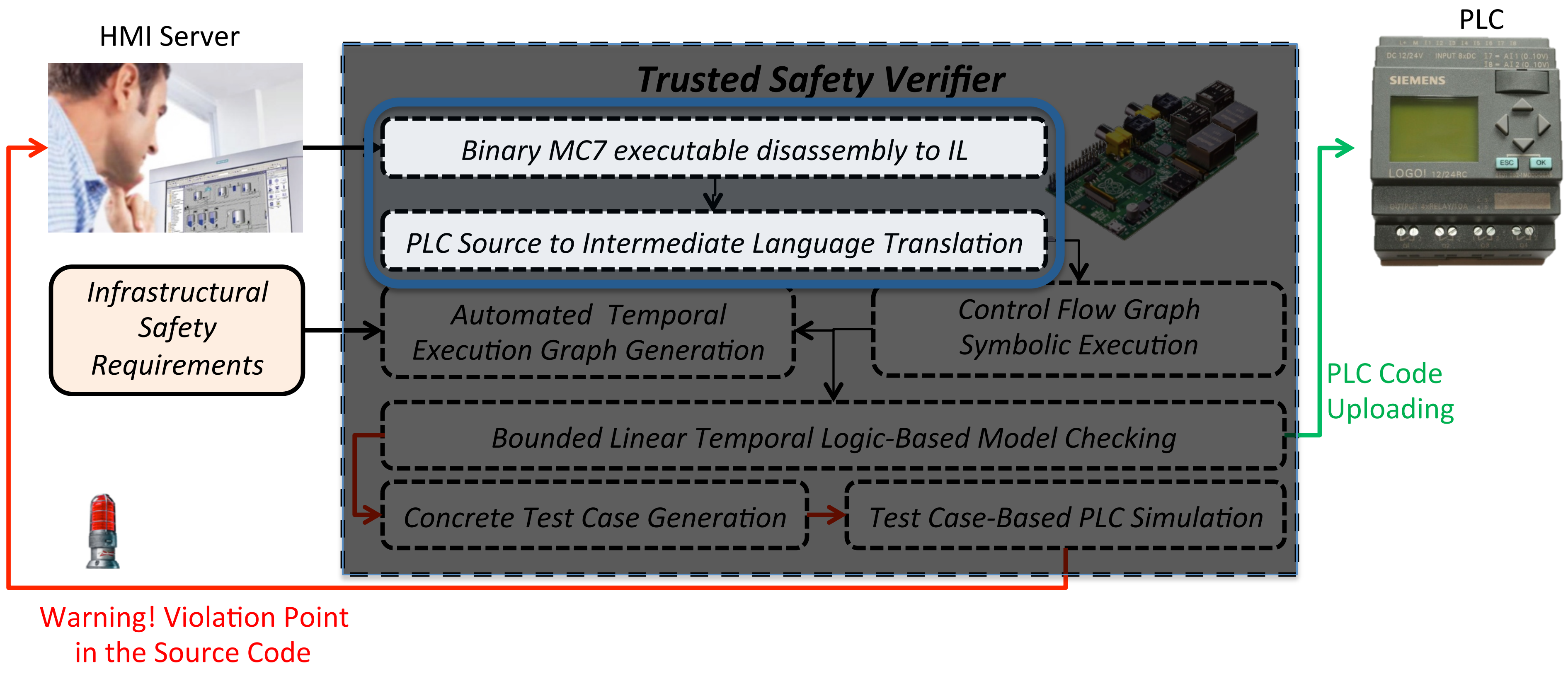


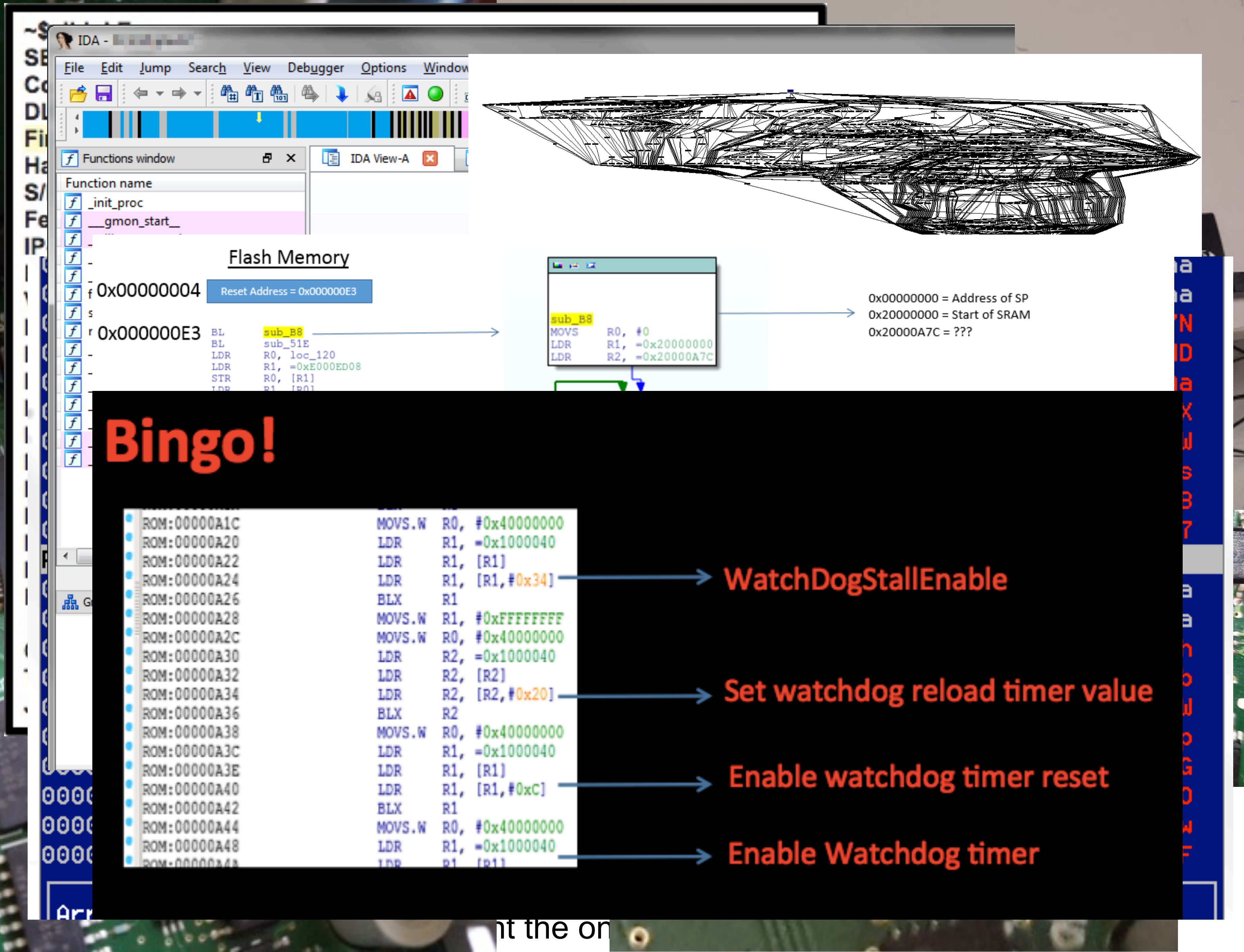
Zonouz et al., Network and Distributed System Security (NDSS)'14;
IEEE S&P Magazine'14; IEEE SmartGridComm'15; IEEE Transactions on SmartGrid'15

Solution Overview


HMI Server
PLC
Trusted Safety Verifier
~~Binary MC7 executable disassembly to IL~~
PLC Source to Intermediate Language Translation
Infrastructural Safety Requirements
Automated Temporal Execution Graph Generation
Control Flow Graph Symbolic Execution
Bounded Linear Temporal Logic-Based Model Checking
Concrete Test Case Generation
Test Case-Based PLC Simulation
PLC Code Uploading
Warning! Violation Point in the Source Code

Solution Overview


HMI Server
Trusted Safety Verifier
Binary MC7 executable disassembly to IL
PLC Source to Intermediate Language Translation
Infrastructural Safety Requirements
Automated Temporal Execution Graph Generation
Control Flow Graph Symbolic Execution
Bounded Linear Temporal Logic-Based Model Checking
Concrete Test Case Generation
Test Case-Based PLC Simulation
PLC
PLC Code Uploading
Warning! Violation Point in the Source Code



Flash Memory

0x00000004 Reset Address = 0x000000E3

0x000000E3
```
BL      sub_B8
BL      sub_51E
LDR     R0, loc_120
LDR     R1, =0xE000ED08
STR     R0, [R1]
```

```
sub_B8
MOVS    R0, #0
LDR     R1, =0x20000000
LDR     R2, =0x20000A7C
```

0x00000000 = Address of SP
0x20000000 = Start of SRAM
0x20000A7C = ???

Bingo!

```
ROM:00000A1C        MOVS.W  R0, #0x40000000
ROM:00000A20        LDR     R1, =0x1000040
ROM:00000A22        LDR     R1, [R1]
ROM:00000A24        LDR     R1, [R1,#0x34]
ROM:00000A26        BLX     R1
ROM:00000A28        MOVS.W  R1, #0xFFFFFFFF
ROM:00000A2C        MOVS.W  R0, #0x40000000
ROM:00000A30        LDR     R2, =0x1000040
ROM:00000A32        LDR     R2, [R2]
ROM:00000A34        LDR     R2, [R2,#0x20]
ROM:00000A36        BLX     R2
ROM:00000A38        MOVS.W  R0, #0x40000000
ROM:00000A3C        LDR     R1, =0x1000040
ROM:00000A3E        LDR     R1, [R1]
ROM:00000A40        LDR     R1, [R1,#0xC]
ROM:00000A42        BLX     R1
ROM:00000A44        MOVS.W  R0, #0x40000000
ROM:00000A48        LDR     R1, =0x1000040
```

WatchDogStallEnable

Set watchdog reload timer value

Enable watchdog timer reset

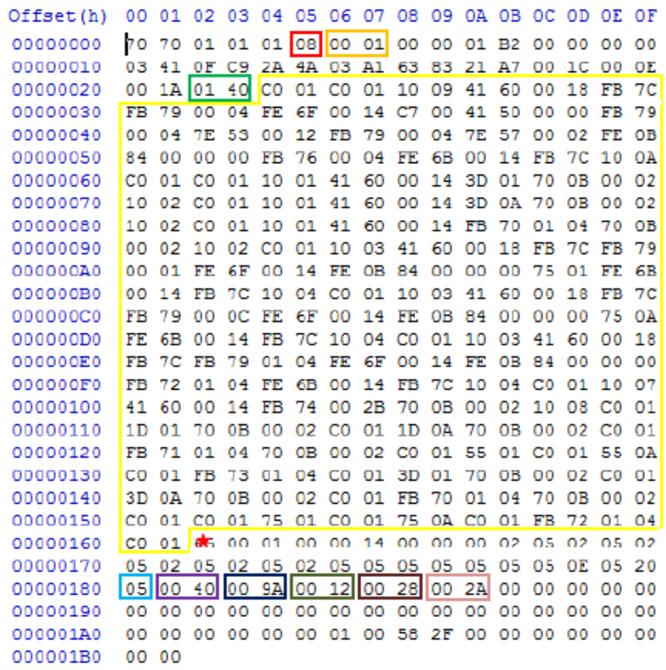
Enable Watchdog timer

nt the on

Binary Executable Disassembly

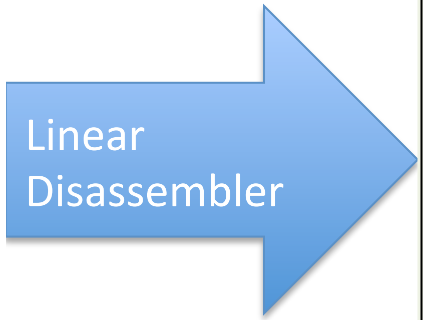
Binary format (MC7):
Siemens Machine Code (.mc7)

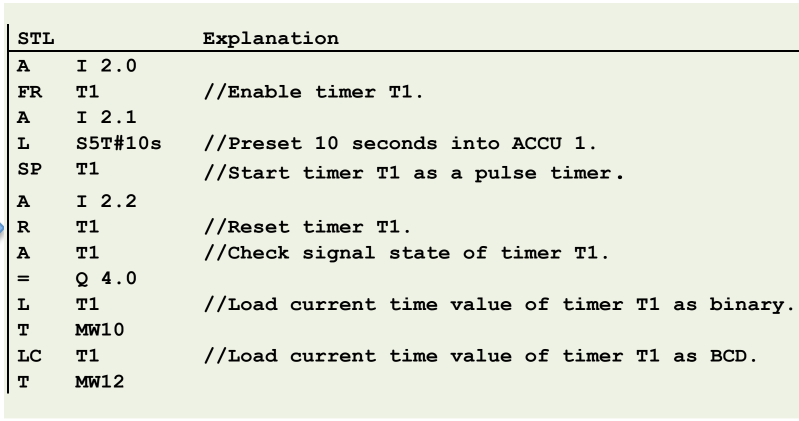
Source code (IL):
Siemens Instruction List (.il)



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  70 70 01 01 01 08 00 01 00 00 01 B2 00 00 00 00
00000010  03 41 0F C9 2A 4A 03 A1 63 83 21 A7 00 1C 00 0E
00000020  00 1A 01 40 C0 01 C0 01 10 09 41 60 00 18 FB 7C
00000030  FB 79 00 04 FE 6F 00 14 C7 00 41 50 00 00 FB 79
00000040  00 04 7E 53 00 12 FB 79 00 04 7E 57 00 02 FE 0B
00000050  84 00 00 00 FB 76 00 04 FE 6B 00 14 FB 7C 10 0A
00000060  C0 01 C0 01 10 01 41 60 00 14 3D 01 70 0B 00 02
00000070  10 02 C0 01 10 01 41 60 00 14 3D 0A 70 0B 00 02
00000080  10 02 C0 01 10 01 41 60 00 14 FB 70 01 04 70 0B
00000090  00 02 10 02 C0 01 10 03 41 60 00 18 FB 7C FB 79
000000A0  00 01 FE 6F 00 14 FE 0B 84 00 00 00 75 01 FE 6B
000000B0  00 14 FB 7C 10 04 C0 01 10 03 41 60 00 18 FB 7C
000000C0  FB 79 00 0C FE 6F 00 14 FE 0B 84 00 00 00 75 0A
000000D0  FE 6B 00 14 FB 7C 10 04 C0 01 10 03 41 60 00 18
000000E0  FB 7C FB 79 01 04 FE 6F 00 14 FE 0B 84 00 00 00
000000F0  FB 72 01 04 FE 6B 00 14 FB 7C 10 04 C0 01 10 07
00000100  41 60 00 14 FB 74 00 2B 70 0B 00 02 10 08 C0 01
00000110  1D 01 70 0B 00 02 C0 01 1D 0A 70 0B 00 02 C0 01
00000120  FB 71 01 04 70 0B 00 02 C0 01 55 01 C0 01 55 0A
00000130  C0 01 FB 73 01 04 C0 01 3D 01 70 0B 00 02 C0 01
00000140  3D 0A 70 0B 00 02 C0 01 FB 70 01 04 70 0B 00 02
00000150  C0 01 C0 01 75 01 C0 01 75 0A C0 01 FB 72 01 04
00000160  C0 01 65 00 01 00 00 14 00 00 00 02 05 02 05 02
00000170  05 02 05 02 05 02 05 05 05 05 05 05 05 0E 05 20
00000180  05 00 40 00 9A 00 12 00 28 00 2A 00 00 00 00 00
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A0  00 00 00 00 00 00 01 00 58 2F 00 00 00 00 00 00
000001B0  00 00
```

🟥 Type of Block (OB,FC,FB,SFC,SFB) (always @ 0x05)
🟧 Block Number (always @ 0x06 and 0x07)
🟩 (Size of Code + 0x02) (always @ 0x22 and 0x23)
🟨 Code
🟦 Number of Network
🟪 Size of Network n°1
🟦 Size of Network n°2
🟩 Size of Network n°3
🟫 Size of Network n°4
🟥 Size of Network n°5





```
STL              Explanation
A    I 2.0
FR   T1          //Enable timer T1.
A    I 2.1
L    S5T#10s     //Preset 10 seconds into ACCU 1.
SP   T1          //Start timer T1 as a pulse timer.
A    I 2.2
R    T1          //Reset timer T1.
A    T1          //Check signal state of timer T1.
=    Q 4.0
L    T1          //Load current time value of timer T1 as binary.
T    MW10
LC   T1          //Load current time value of timer T1 as BCD.
T    MW12
```

PLC Architecture

- Hierarchical addresses
  - *Not just integers! Prefixed namespace qualifiers*
  - *Siemens/Allen-Bradley has 1/3 qualifiers*
- Multi-indexed memory
  - *Multiple-size memory*
  - *Addressing word- byte- and bit-level*

IL Source Code Lifting

Source code (IL):
Siemens Instruction List (.il)

STL		Explanation
A	I 2.0	
FR	T1	//Enable timer T1.
A	I 2.1	
L	S5T#10s	//Preset 10 seconds into ACCU 1.
SP	T1	//Start timer T1 as a pulse timer.
A	I 2.2	
R	T1	//Reset timer T1.
A	T1	//Check signal state of timer T1.
=	Q 4.0	
L	T1	//Load current time value of timer T1 as binary.
T	MW10	
LC	T1	//Load current time value of timer T1 as BCD.
T	MW12	

PLC Code Lifter

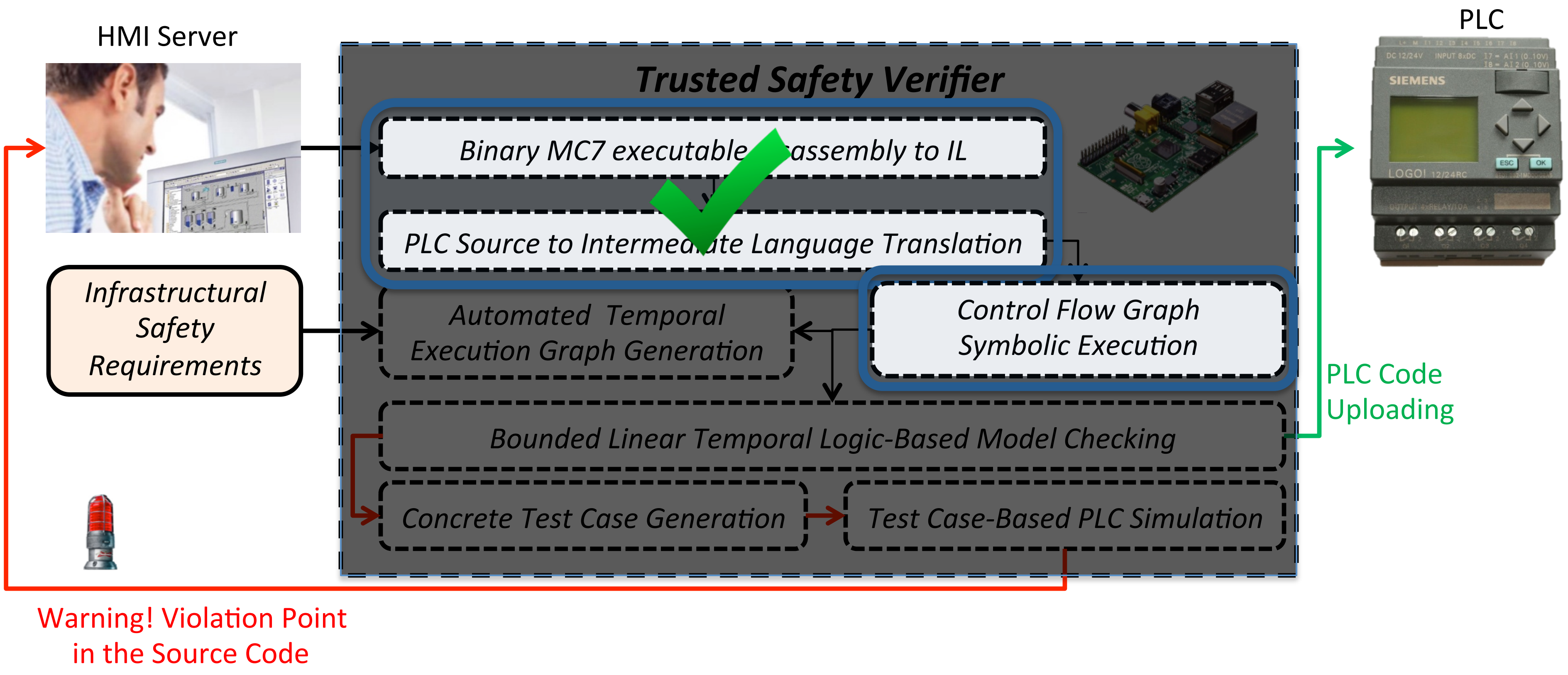
Intermediate level code (ILIL):
Instruction List Intermediate language (.ilil)

```
// Initialize PLC state.
mem := {} : mem_t(1);              // Main memory.
I   := 0;                          // Input memory qualifier.
Q   := 1;                          // Output memory qualifier.
RLO := 1 : reg1_t;                 // Boolean accumulator.
FC  := 0 : reg1_t;                 // System status registers.
STA := 0 : reg1_t;
...

// A I 0.5
STA := load(mem, [I::0::0::0::5]);
cjmp FC == 0 : reg1_t,L1,L2;
label L1;
RLO := STA;
label L2;
RLO := RLO && STA;
FC  := 1 : reg1_t;                 // Side effects.
...

// = Q 0.1
STA := RLO;
mem := store(mem, [Q::0::0::0::1], RLO);
FC  := 0 : reg1_t;                 // Side effects.
...
```

Why "IL → ILIL"?

- Direct IL analysis is difficult
  - *IL syntax/semantics vary widely by vendor*
  - *IL instructions have side affects obscuring certain control flows*
- ILIL: based on the Vine intermediate language
  - *Memory, register and address types*

Solution Overview



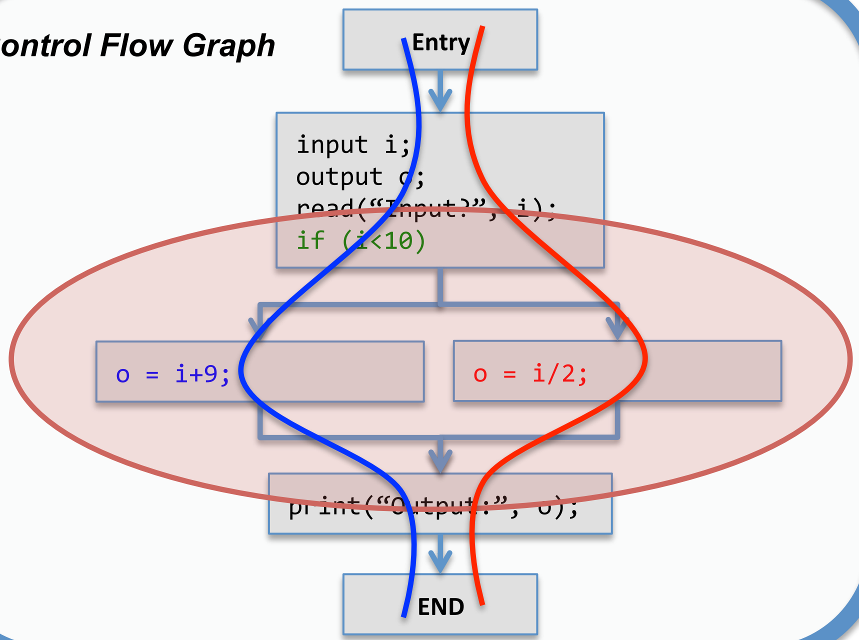
HMI Server
PLC
Trusted Safety Verifier
Binary MC7 executable disassembly to IL
PLC Source to Intermediate Language Translation
Infrastructural Safety Requirements
Automated Temporal Execution Graph Generation
Control Flow Graph Symbolic Execution
Bounded Linear Temporal Logic-Based Model Checking
Concrete Test Case Generation
Test Case-Based PLC Simulation
PLC Code Uploading
Warning! Violation Point in the Source Code

Symbolic Execution Review

Code Segment

```
input i;
output o;
if (i<10)
      o = i+9;
else
      o = i/2;
```

Control Flow Graph



Concrete execution

```
Input? ← 16
Output: 8
```

Symbolic execution

```
Input? ← 'a'
Output:
      PC [a<10 ] → a+9
      PC [a>=10] → a/2
```

ILIL Symbolic Execution

Intermediate level code (ILIL):
Instruction List Intermediate language (.ilil)

```
// Initialize PLC state.
mem := {} : mem_t(1);          // Main memory.
I   := 0;                      // Input memory qualifier.
Q   := 1;                      // Output memory qualifier.
RLO := 1 : reg1_t;             // Boolean accumulator.
FC  := 0 : reg1_t;             // System status registers.
STA := 0 : reg1_t;
...

// A I 0.5
STA := load(mem, [I::0::0::0::5]);
cjmp FC == 0 : reg1_t,L1,L2;
label L1;
RLO := STA;
label L2;
RLO := RLO && STA;
FC  := 1 : reg1_t;             // Side effects.
...

// = Q 0.1
STA := RLO;
mem := store(mem, [Q::0::0::0::1], RLO);
FC  := 0 : reg1_t;             // Side effects.
...
```

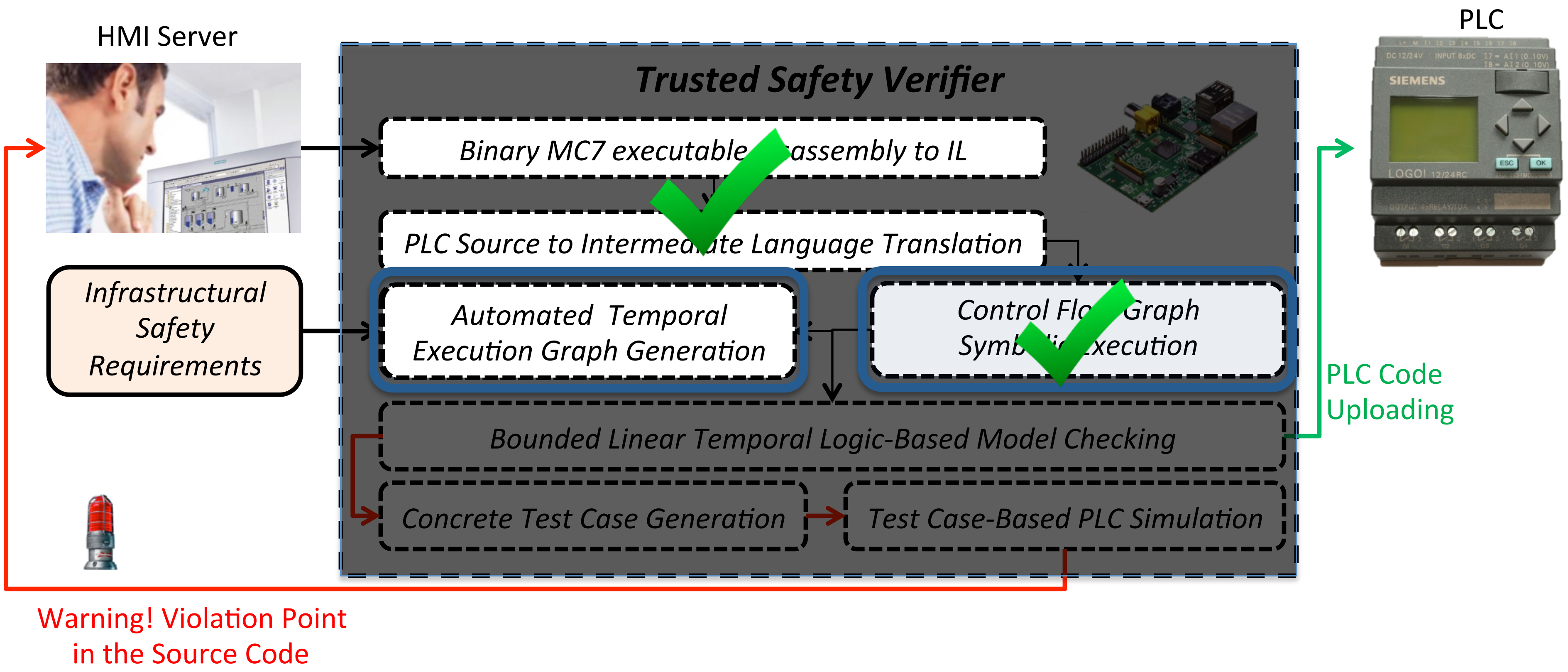
SE Engine

Symbolic Scan Cycle:
(Path condition, symbolic output)

```
Input ← 'a'
Output:
    PC [a<10 ] → a+9
    PC [a>=10] → a/2
```

- Control flow graph exploration
  - *SMT solver for predicate feasibility checking*
  - *E.g., is "(a<10) && (20<a)" satisfiable?*
- Optimizations
  - *Bounded loop execution (guaranteed correctness)*
  - *Indirect function calls w/ symbolic address values*
  - *Opcode-based register type inference*
- Symbolic execution covers a "**single**" scan cycle
  - *Subsequent IO scans are treated independently*
  - *No temporal considerations, e.g., timers/counters*

Solution Overview

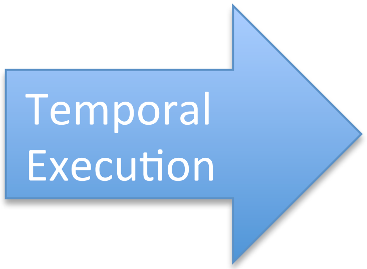


HMI Server
Trusted Safety Verifier
Binary MC7 executable Disassembly to IL
PLC Source to Intermediate Language Translation
Infrastructural Safety Requirements
Automated Temporal Execution Graph Generation
Control Flow Graph Symbolic Execution
Bounded Linear Temporal Logic-Based Model Checking
Concrete Test Case Generation
Test Case-Based PLC Simulation
PLC
SIEMENS
LOGO!
PLC Code Uploading
Warning! Violation Point in the Source Code

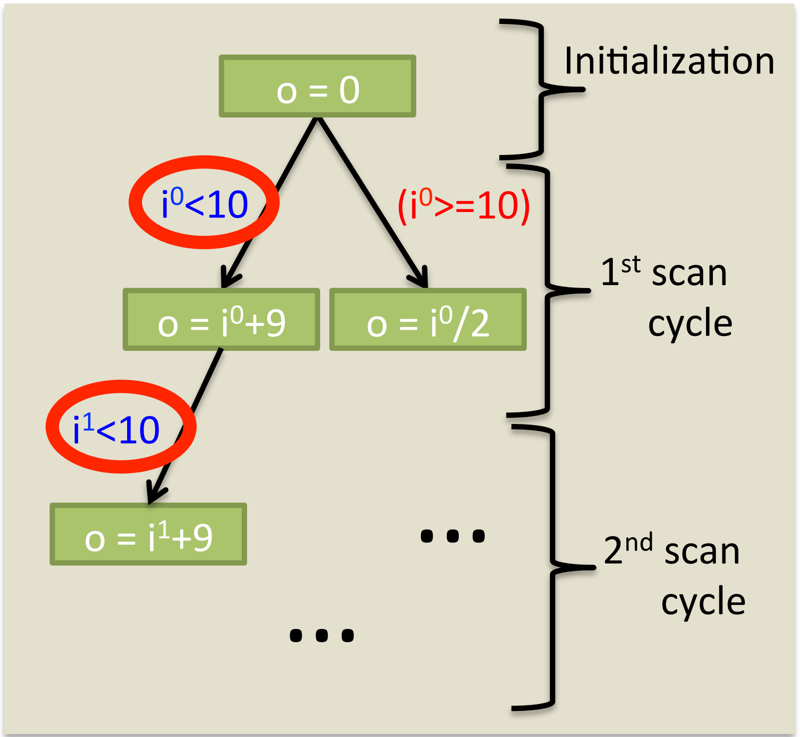
Temporal Execution Graph

Symbolic Scan Cycle:
(Path condition, symbolic output)

```
    Input ← ‘a’
    Output:
          PC [a<10 ] → a+9
          PC [a>=10] → a/2
```

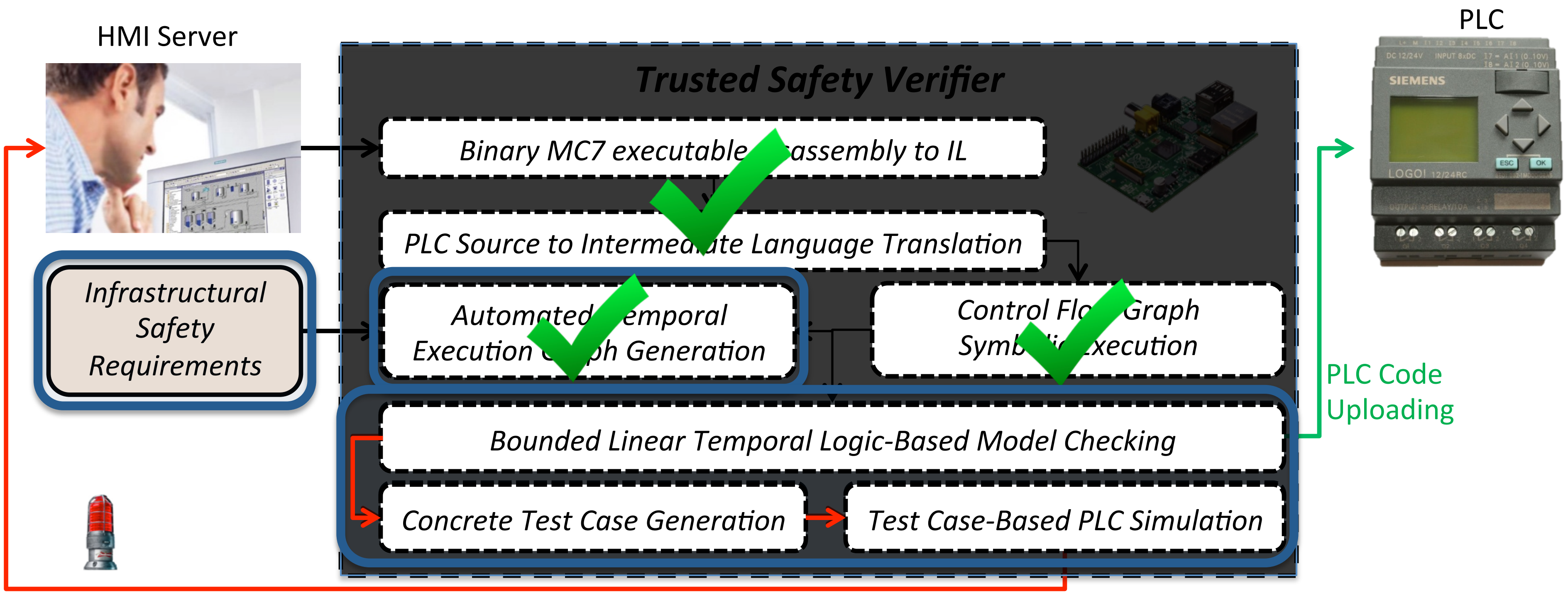


Temporal Execution Graph (TEG):
State notion: symbolic variable vales



- State based model
  - *Transitions denote new cycles*
  - *Each state store symbolic memory values*
- TEG captures inter-cycle dependencies
  - *E.g., timers and counters*
- Recursive TEG generation return conditions
  - *Ideal: all states are generated*
  - *Bounded: finite scan-cycle exploration*

Solution Overview



HMI Server
PLC
Trusted Safety Verifier
Binary MC7 executable Disassembly to IL
PLC Source to Intermediate Language Translation
Infrastructural Safety Requirements
Automated Temporal Execution Graph Generation
Control Flow Graph Symbolic Execution
Bounded Linear Temporal Logic-Based Model Checking
Concrete Test Case Generation
Test Case-Based PLC Simulation
PLC Code Uploading

Warning! Violation Point in the Source Code

Infrastructural Safety Requirements

- Formulated using linear temporal logic expressions
- Example safety requirement
  - *English expression*
    - *Relay R_1 should **NOT** open **UNTIL** Generator G_2 turns on*
  - *Logical expression*
    - *Atomic propositions*
      - a_1: "Relay R_1 is open"
      - a_2: "Generator G_2 is on"

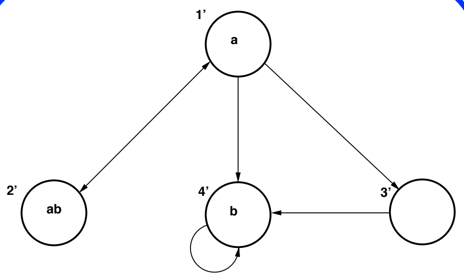
LTL: $!a_1 \mathbf{U} a_2$

Formal Verification

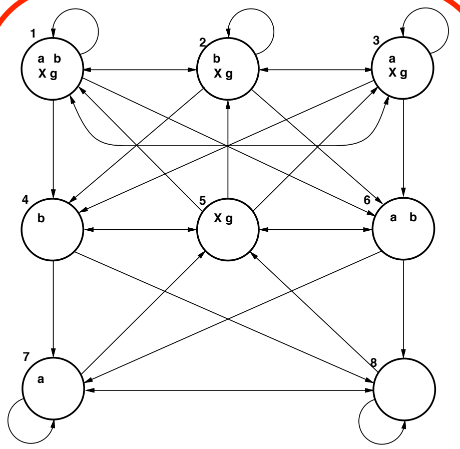
1. Negate the LTL Spec formula
2. Generates the TEG-UR product model P
3. Search for a path in P
4. Get concrete input values

Safety requirements:
$! (a_1 \mathbf{U} a_2)$

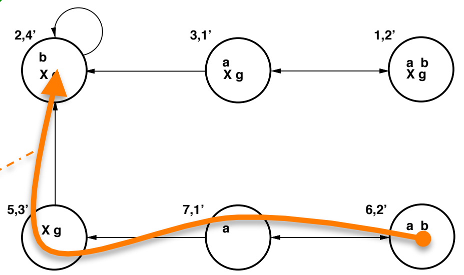
Negation



Temporal Execution Graph **(TEG)**



Unsafety Requirements (**UR**)



P = TEG x UR

Req-violating input vector:

$(i^0, i^1, i^2) = (10, 2, 15)$

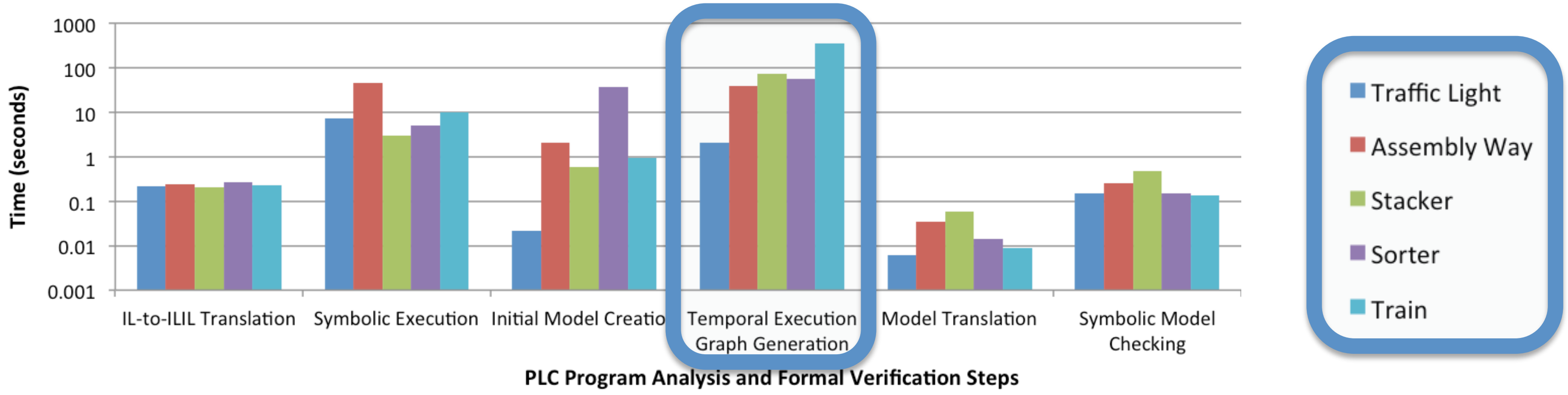
Req-violating path condition:

$(i^0 < 12)$ & $(0 < i^1)$ & $(i^2 < 20)$

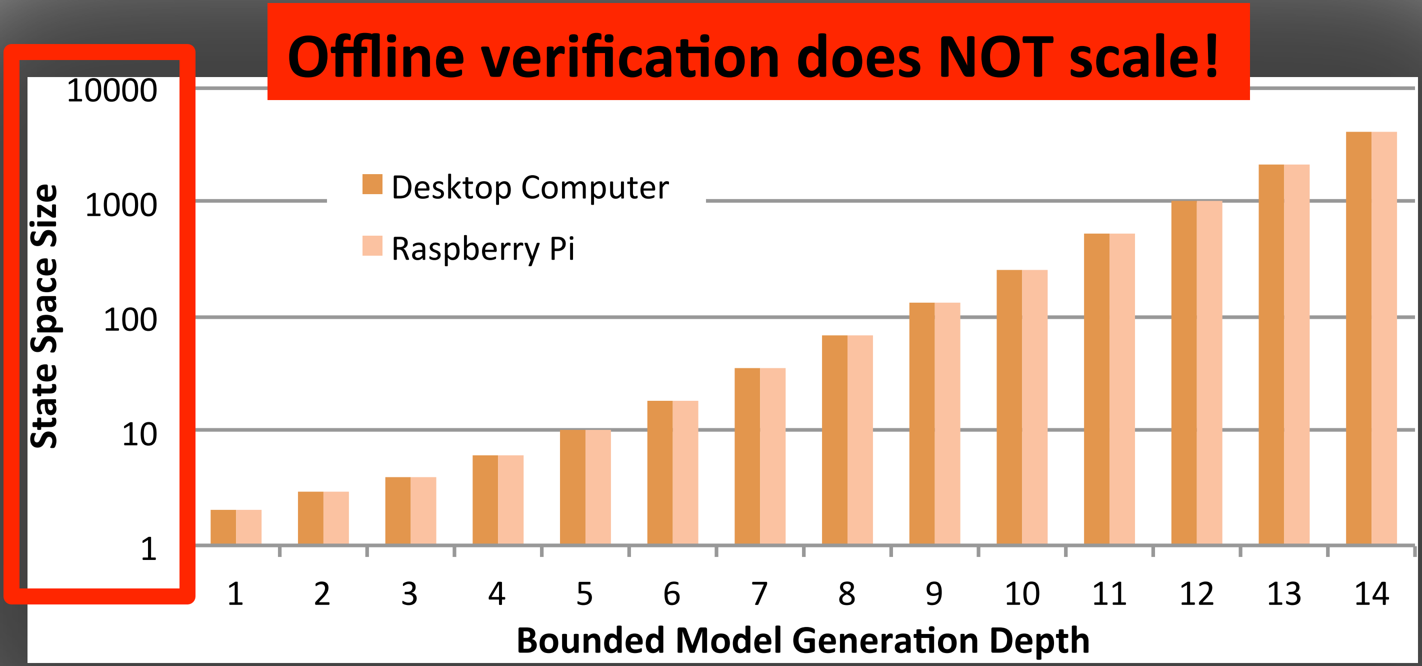
Evaluations

- Implementations
  - *>30K LOC*
  - *C/C++*
- Experimental setup
  - *Raspberry Pi – Linux 3.2.27*
  - *Desktop – Ubuntu; 8 GB RAM; Linux 3.5*
- Case studies
  - *Five real Siemens PLC controller programs*
    - *Traffic light, Rail Inter-locking, Assembly line, Stacker, Sorter*

Detailed Performance Analysis



Practical Feasibility



Practical Solution?

Past Work

Offline formal verification and model checking

- Unscalable for large-scale platforms

Runtime monitoring and intrusion detection

- Too late for effective response and recovery

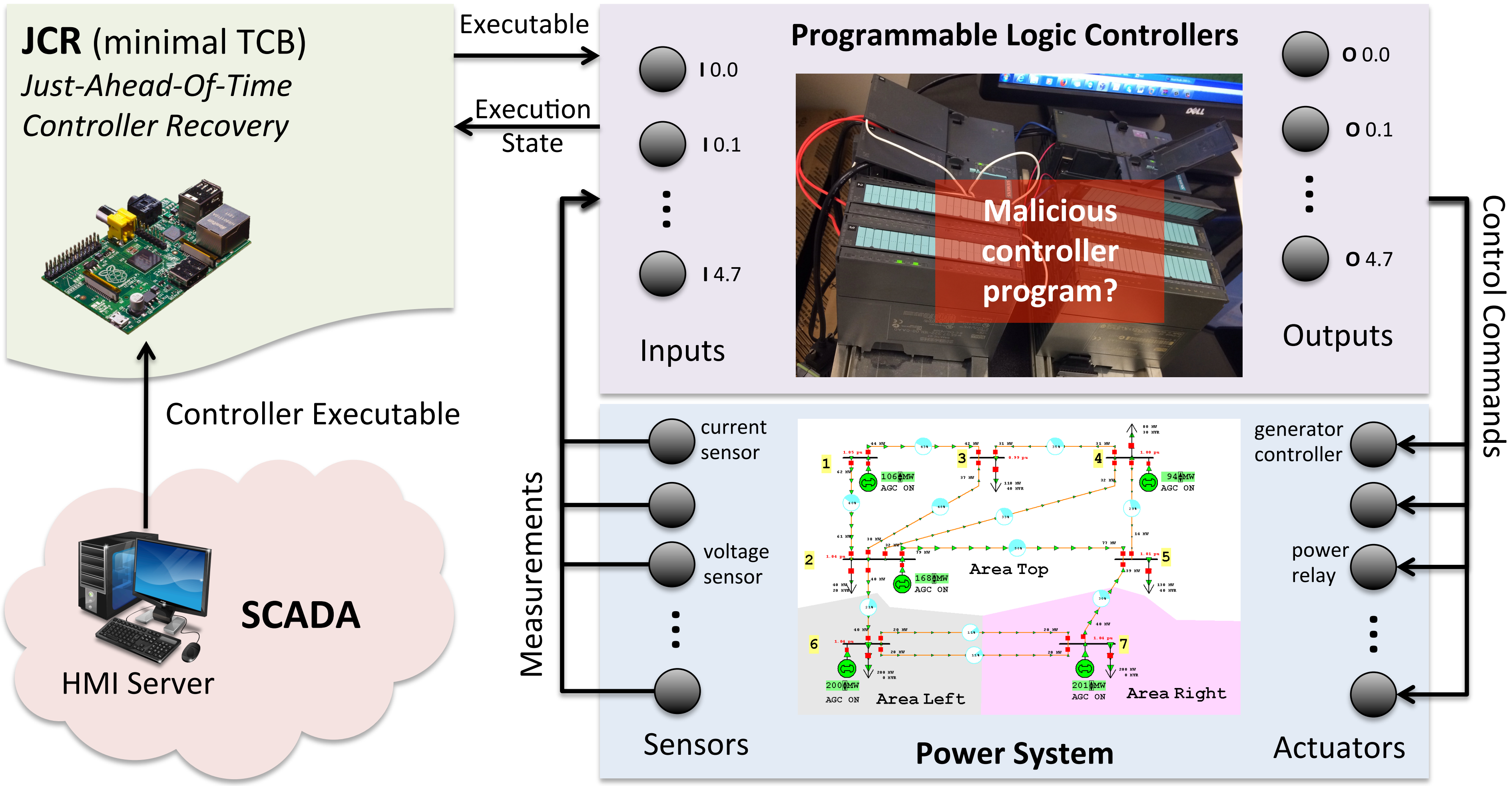
Our Solution

Just-Ahead-Of-Time Verification and Response

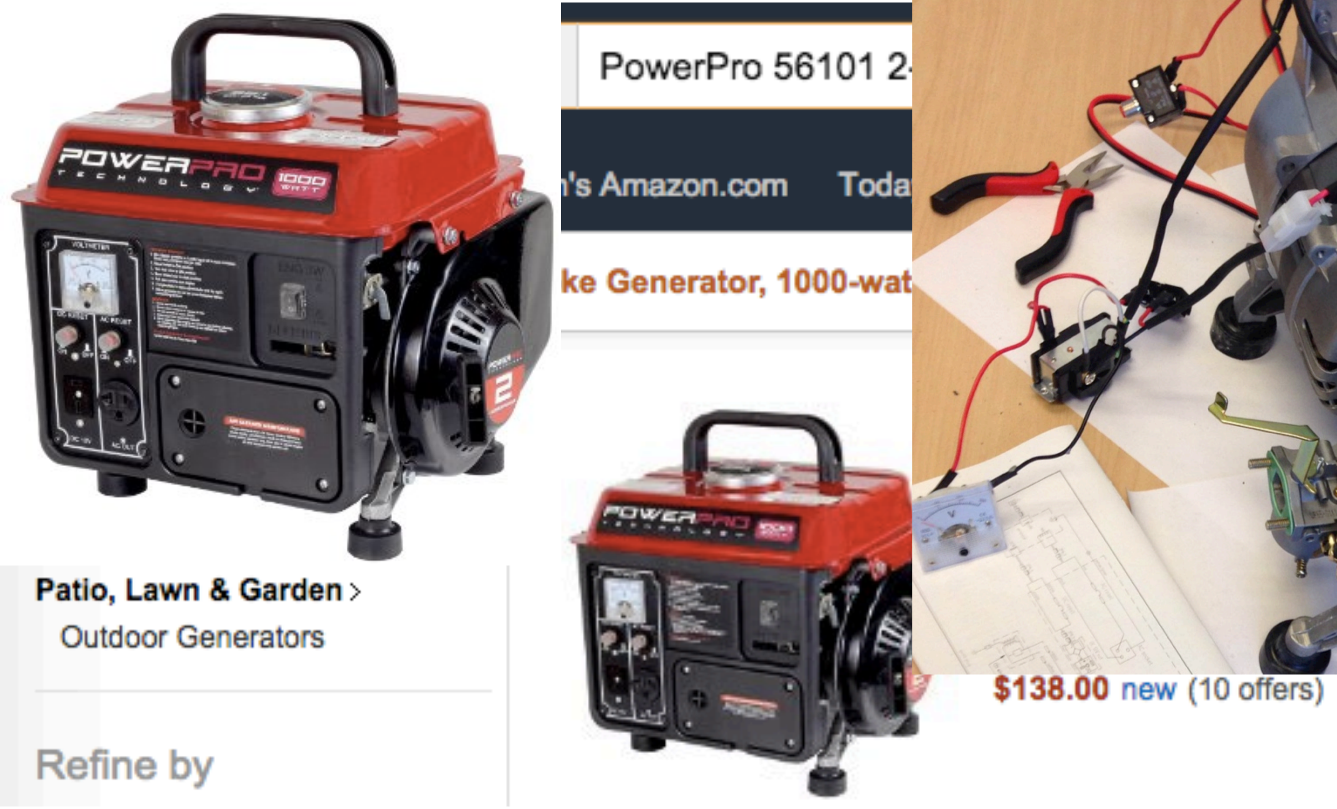
+ Remarkably smaller system models to analyze

+ Sufficient time for timely intrusion tolerance

Just-Ahead-Of-Time Verification



Generator Reverse Engineering



PowerPro 56101 2-

's Amazon.com Toda

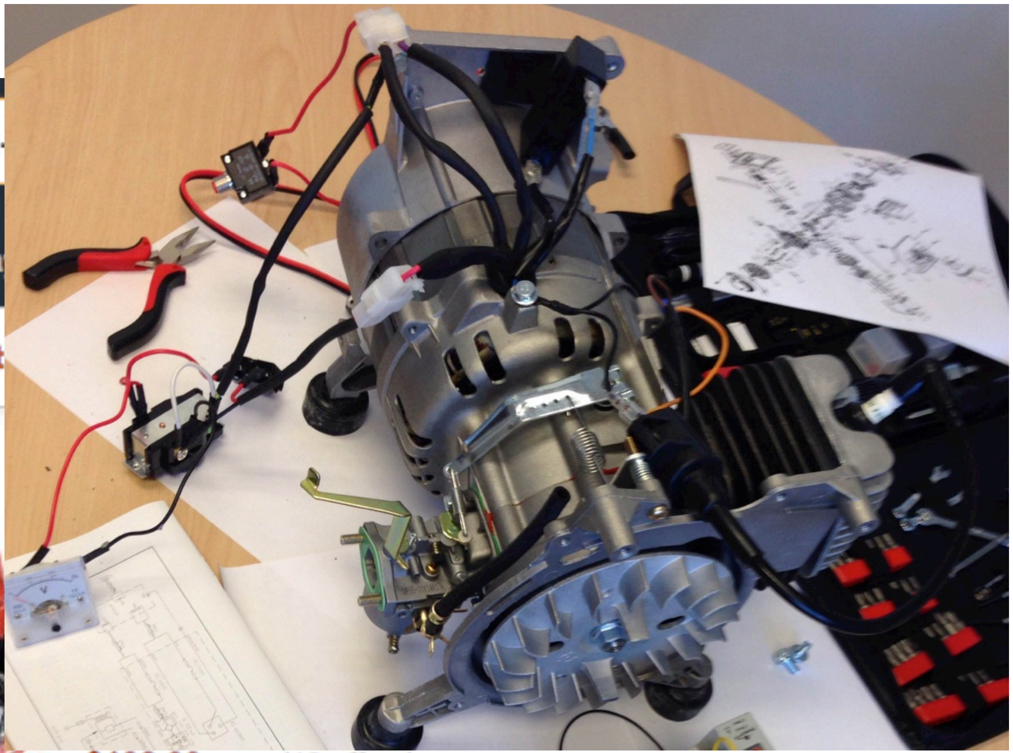
ke Generator, 1000-wat

Patio, Lawn & Garden ›

Outdoor Generators

Refine by

$138.00 new (10 offers)



Acknowledgements

- Collaborators

PSU UF UNIVERSITY of FLORIDA I MIT Ucla R

SIEMENS AT&T intel

- Sponsors

DEPARTMENT OF THE NAVY USN ONR Science & Technology NSF DEPARTMENT OF ENERGY UNITED STATES OF AMERICA U.S. DEPARTMENT OF HOMELAND SECURITY

arpa·e FORTINET Google

WIND RIVER GRAMMATECH

Conclusions

- **Optimal control vs. safety redlines**
  - *reject the **control** that **violate** the power system **safety** requirements*
  - *replace them with security/safety-preserving **countermeasures***
- **Minimal** trusted computing base for infrastructural **resilience**
  - ***easier to analyze**, verify its correctness, and **protect** its cyber-security*
  - *guarantee **safety** while **"huge"** SCADA solves for the **optimal plant control***
- **JAT** verification allows for countermeasure **selection**
  - ***proactive** tolerance to prevent **too-late responses***
  - ***learns** decided-upon responses for later **similar** unsafe states*

Saman Zonouz saman.zonouz@rutgers.edu **Thank You!**