CPS & IoT Security Research

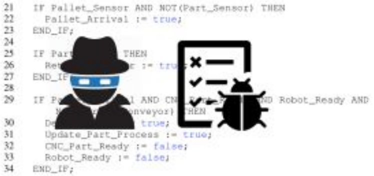
Yue Duan
Illinois Institute of Technology

Towards Automated Safety Vetting of PLC Code in Real-World Plants

Mu Zhang[∗], Chien-Ying Chen[†], Bin-Chou Kao[‡], Yassine Qamsane[§], Yuru Shao[¶], Yikai Lin[¶], Elaine Shi[∗], Sibin Mohan[†], Kira Barton[§], James Moyne[§] and Z. Morley Mao[¶]

[∗]CS, Cornell; [†]CS, UIUC; [‡]ITI, UIUC; [§]ME, UMich; [¶]EECS, UMich

PLC being a Major Attack Vector



```
IF Pallet_Sensor AND NOT(Part_Sensor) THEN
  Pallet_Arrival := true;
END_IF;

IF Part[illegible] THEN
  Ret[illegible]r := tru[illegible]
END_IF

IF Pa[illegible]l AND CN[illegible]ND Robot_Ready AND
    N[illegible]onveyor) THEN
  De[illegible] true;
  Update_Part_Process := true;
  CNC_Part_Ready := false;
  Robot_Ready := false;
END_IF;
```

Controller Code w/ Safety Violations

Insider Attacks or Bugs

Programmable Logic Controller (PLC)

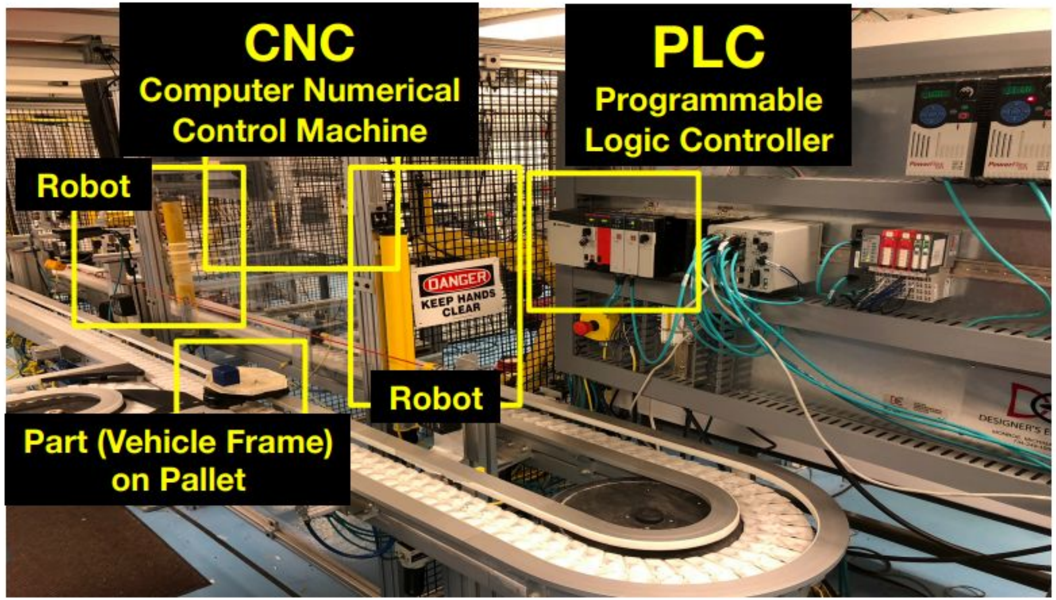
Core Control Unit on the Factory Floor

Physical Damage

Different from Financial Loss Often Seen in Attacks in Consumer Systems

Overlooked Fact:

- ICS is Complex, PLC is NOT Working Alone


CNC
Computer Numerical Control Machine
PLC
Programmable Logic Controller
Robot
Robot
Part (Vehicle Frame) on Pallet
DANGER
KEEP HANDS CLEAR

Real-world Automotive Manufacturing Testbed

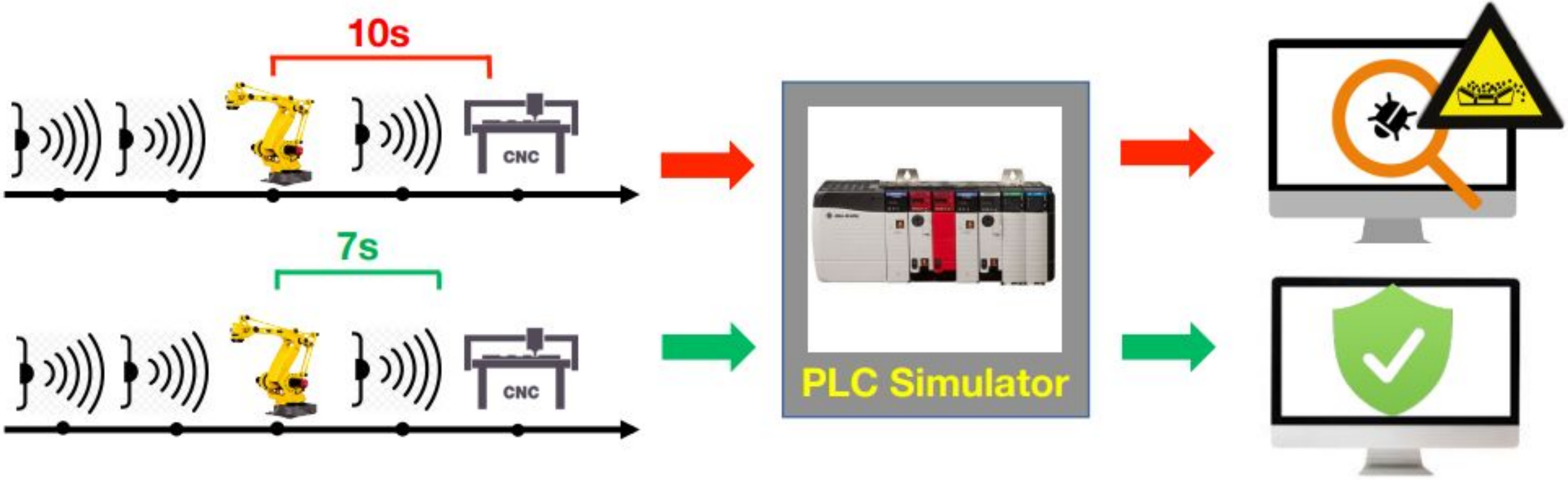
Developed by **No.1 Vendor** (Rockwell Automation)

PLCs are **driven by events** from other machines

Testing PLC code requires external event inputs

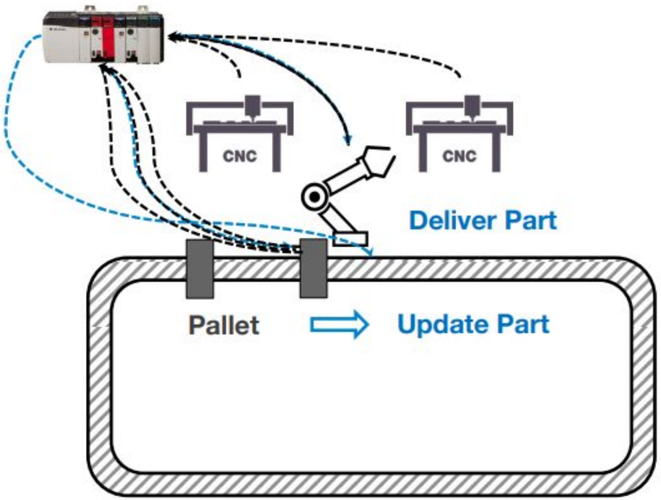
Rearranging Event Order to Test PLC Code

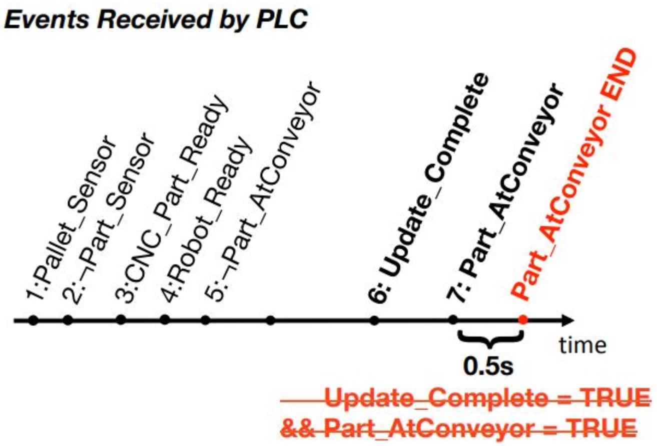
is NOT Sufficient

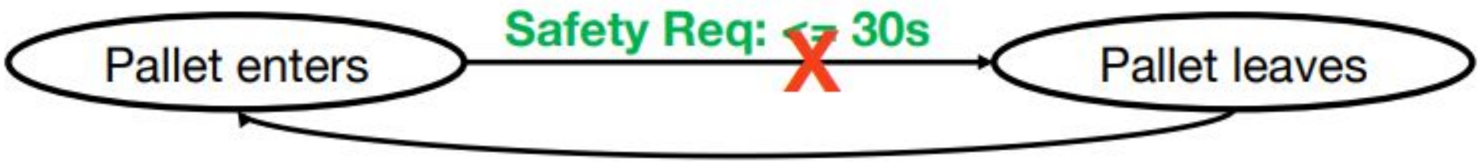


Event Sequences of Same Ordering But Different Timings

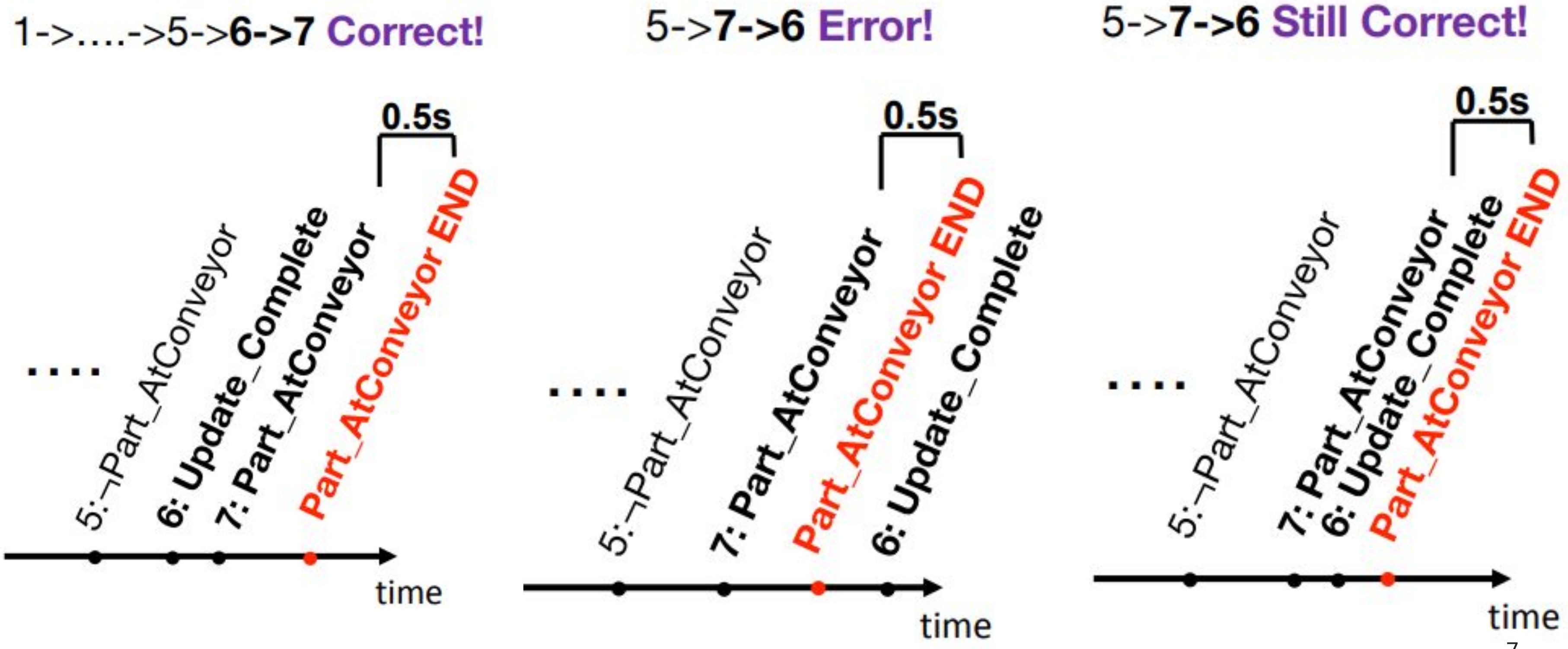
Running Example







Traditional Event Permutation



1->....->5->**6->7** Correct!

0.5s

....

5:¬Part_AtConveyor

6: Update_Complete

7: Part_AtConveyor

Part_AtConveyor END

time

5->**7->6** Error!

0.5s

....

5:¬Part_AtConveyor

7: Part_AtConveyor

Part_AtConveyor END

6: Update_Complete

time

5->**7->6** Still Correct!

0.5s

....

5:¬Part_AtConveyor

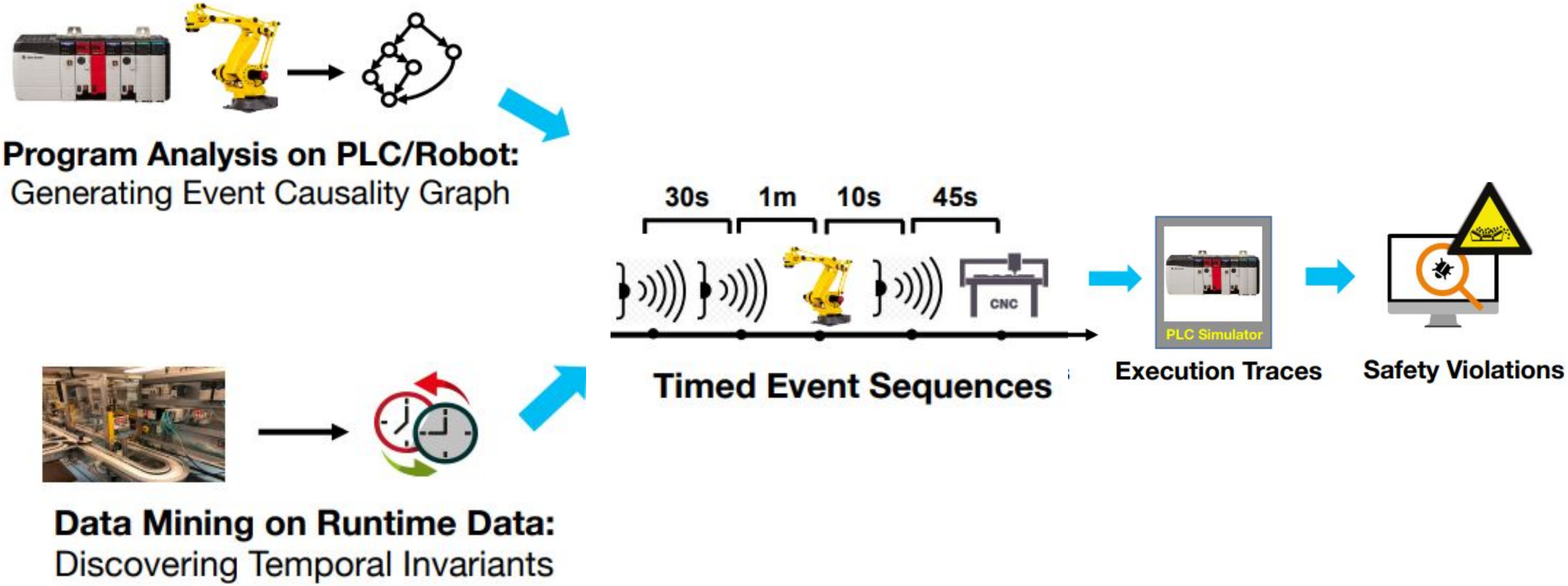
7: Part_AtConveyor

6: Update_Complete

Part_AtConveyor END

time

VETPLC: Generating Timed Event Sequences



Program Analysis on PLC/Robot:
Generating Event Causality Graph

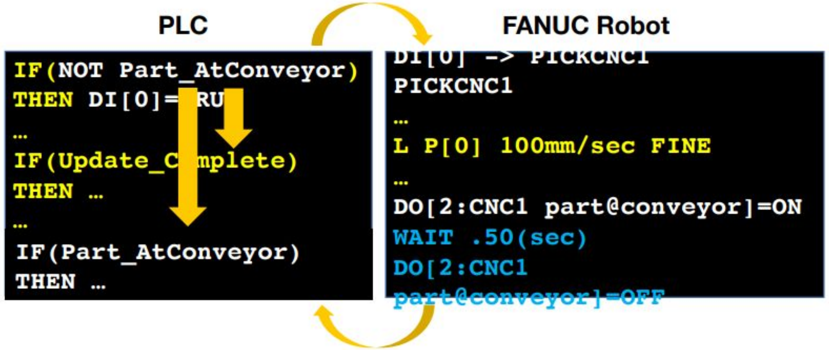
Data Mining on Runtime Data:
Discovering Temporal Invariants

Timed Event Sequences

Execution Traces

Safety Violations

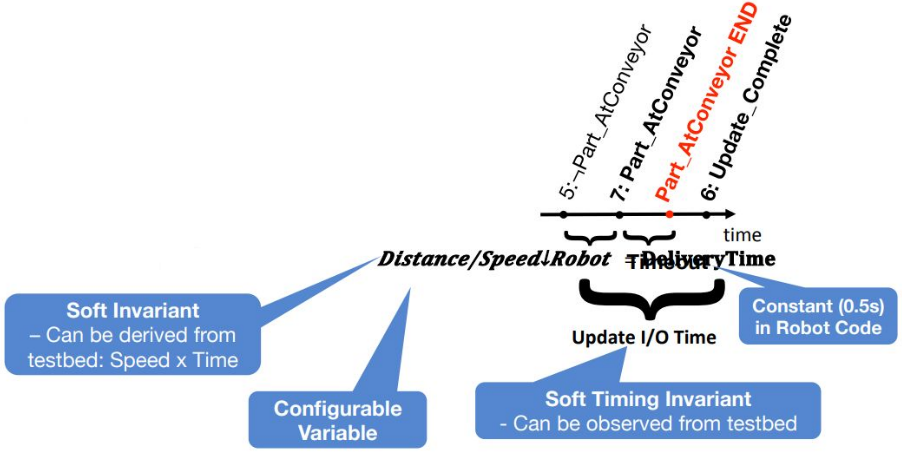
VetPLC


PLC

```
IF(NOT Part_AtConveyor)
THEN DI[0]=TRUE
…
IF(Update_Complete)
THEN …
…
IF(Part_AtConveyor)
THEN …
```

FANUC Robot

```
DI[0] -> PICKCNC1
PICKCNC1
…
L P[0] 100mm/sec FINE
…
DO[2:CNC1 part@conveyor]=ON
WAIT .50(sec)
DO[2:CNC1
part@conveyor]=OFF
```



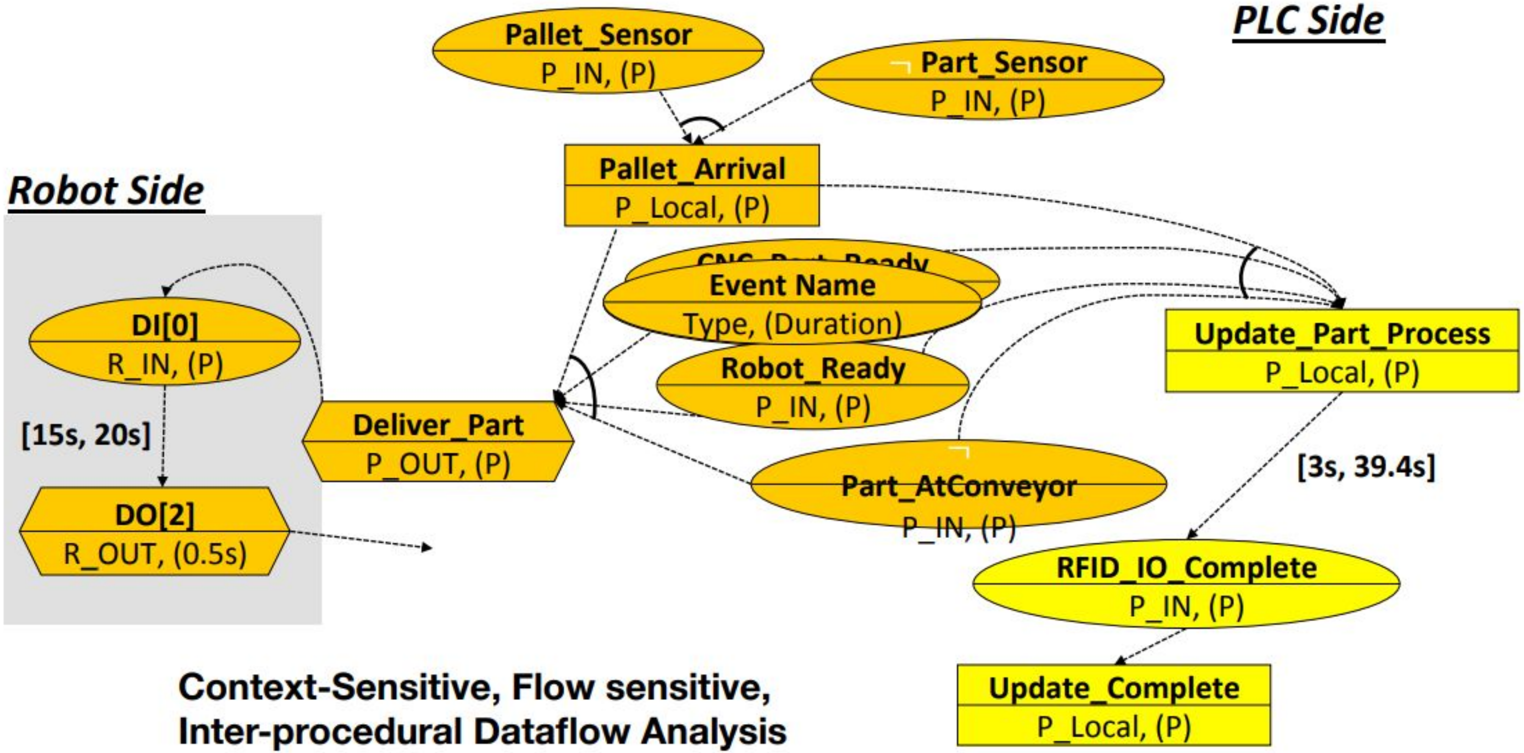
Soft Invariant
– Can be derived from testbed: Speed x Time

Configurable Variable

Soft Timing Invariant
- Can be observed from testbed

Constant (0.5s) in Robot Code

Timed Event Causality Graph (TECG)



Context-Sensitive, Flow sensitive, Inter-procedural Dataflow Analysis

Mining Temporal Invariants for Events: 2 Steps

Step 1: Qualitative “followed-by”:

– Synoptic (*FSE’11*)

$$\textbf{Follows}[\varepsilon_a][\varepsilon_b] = \textbf{Occurrence}[\varepsilon_a]$$

Step 2: Quantitative “with-in”:

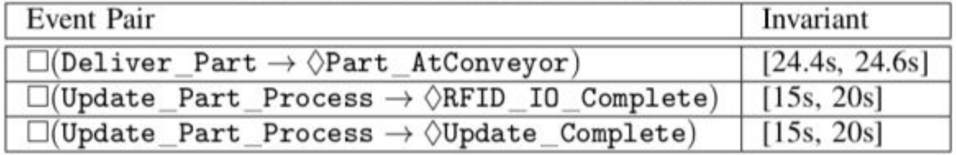
– Perfume (*ASE’14*)

$$\square\, t_x.(\varepsilon_a \rightarrow \Diamond t_y.(\varepsilon_b \wedge t_y - t_x \geq \tau_{lower}))$$

$$\square\, t_x.(\varepsilon_a \rightarrow \Diamond t_y.(\varepsilon_b \wedge t_y - t_x \leq \tau_{upper}))$$

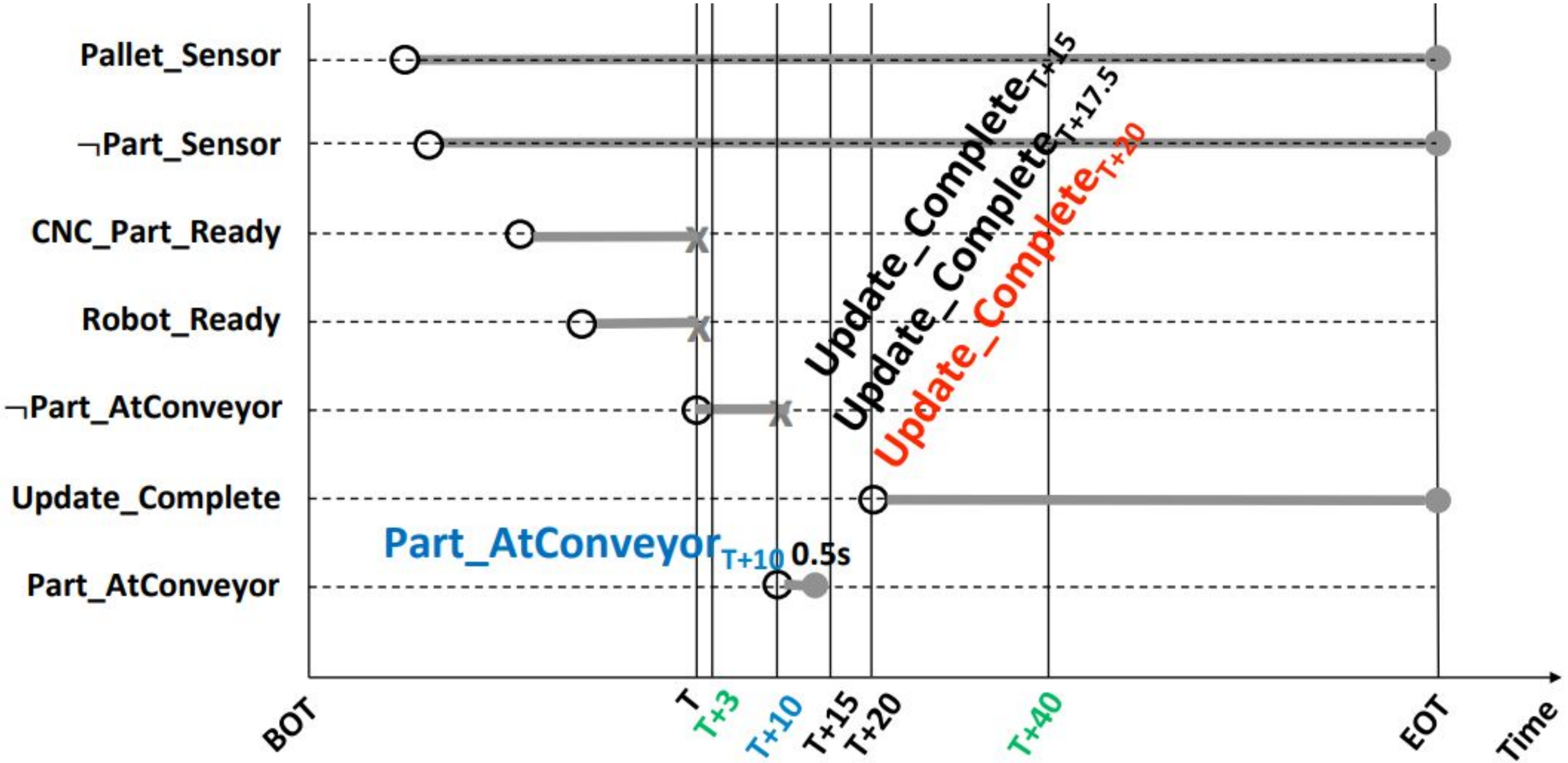
Results for Motivating Example (1.2 GB data for 10 hours):

TABLE I: Mined Invariants



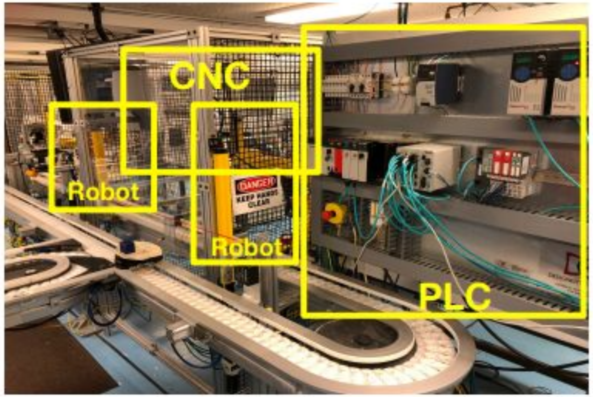
Event Pair	Invariant
□(Deliver_Part → ◊Part_AtConveyor)	[24.4s, 24.6s]
□(Update_Part_Process → ◊RFID_IO_Complete)	[15s, 20s]
□(Update_Part_Process → ◊Update_Complete)	[15s, 20s]

Creating Timed Event Sequences

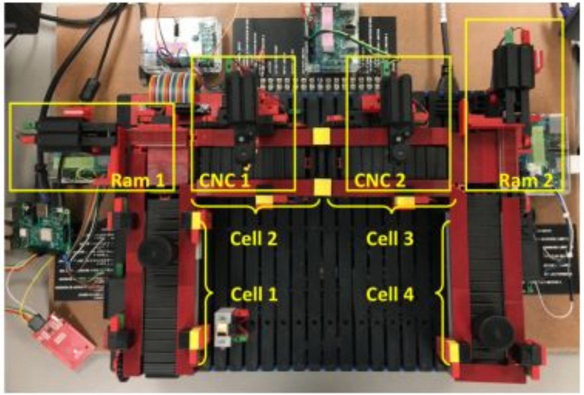


Evaluation on Real Testbeds for Different Scenarios

2 Different Testbeds



***SMART*: Automotive Production Line**



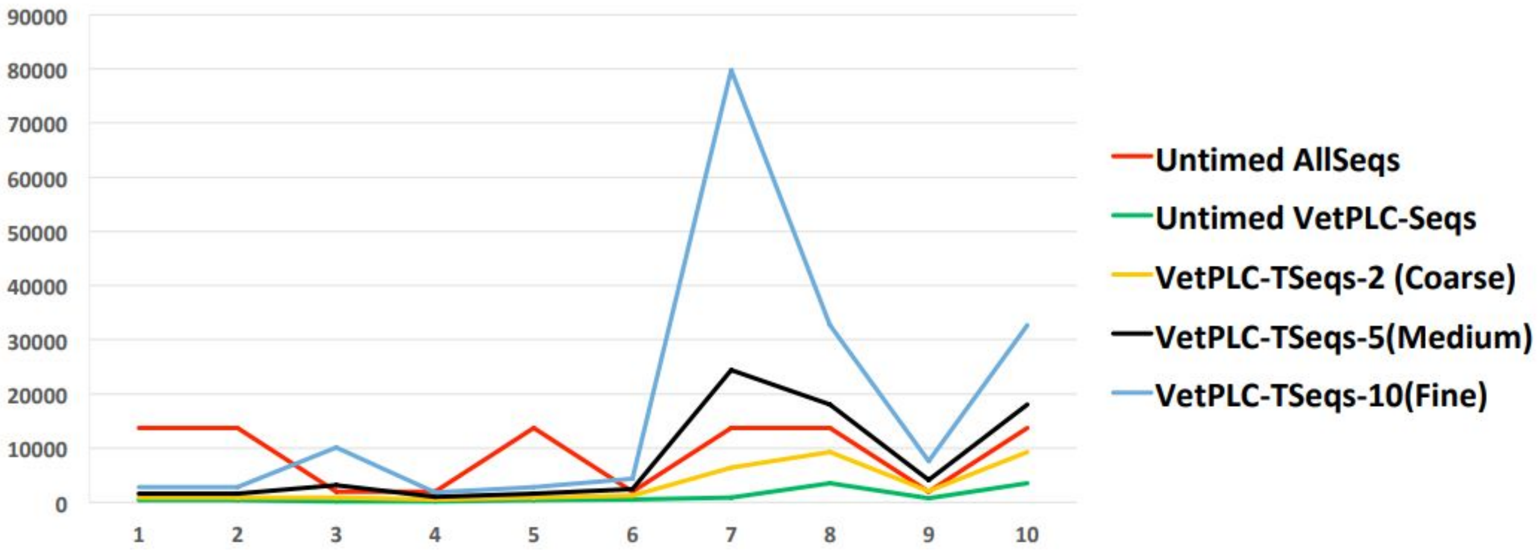
***Fischertechnik*: Part Processing w/ 4 PLCs**

10 Safety-critical Scenarios

S1: Conveyor Overflow #1
S2: Robot in Danger Zone
S3: Conveyor Overflow #2
S4: Part-Gate Collision
S5: CNC Overflow

S6: Ram-Part Collision
S7: CNC-Part Collision
S8: Conveyor Overflow #3
S9: Conveyor Underflow
S10: Ram-Part Collision #2

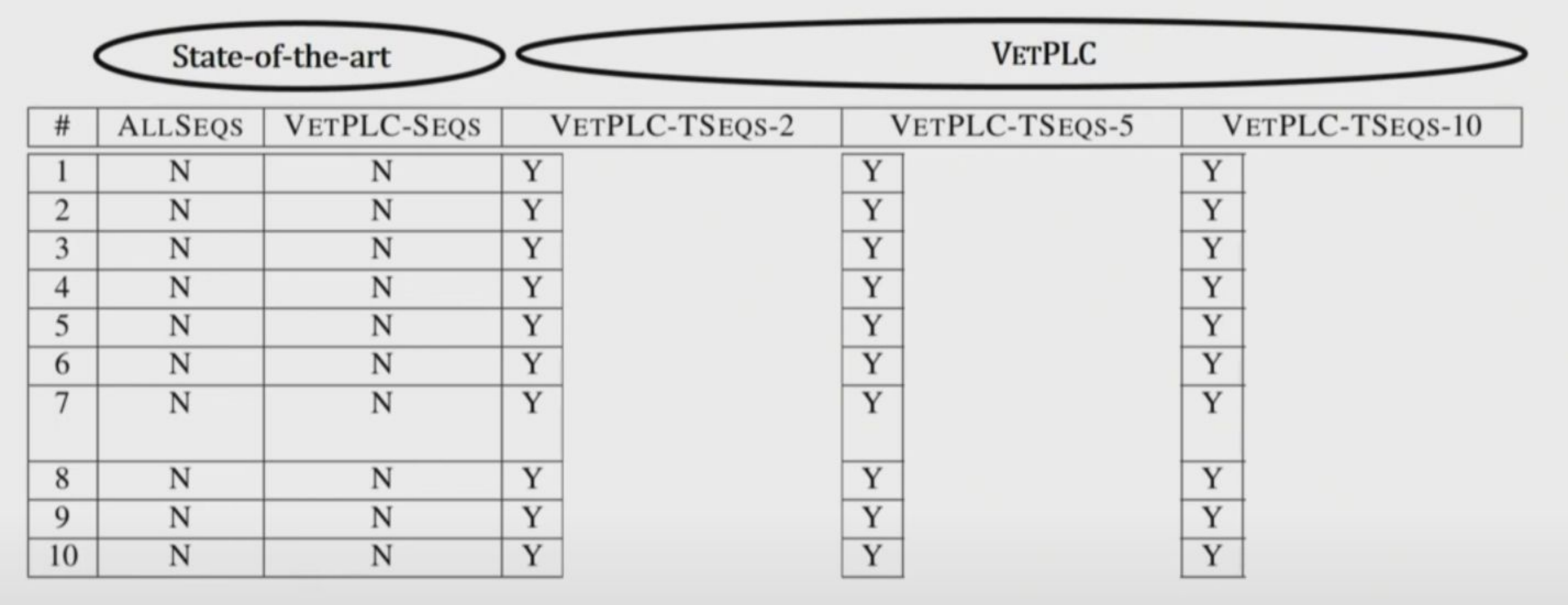
Evaluation: How many sequences are created?



Red → Green: Program analysis reduces amount of event sequences

Green → Orange → Black → Blue: Time discretization can significantly increases that

Bug Detected? State-of-the-Art vs. VETPLC



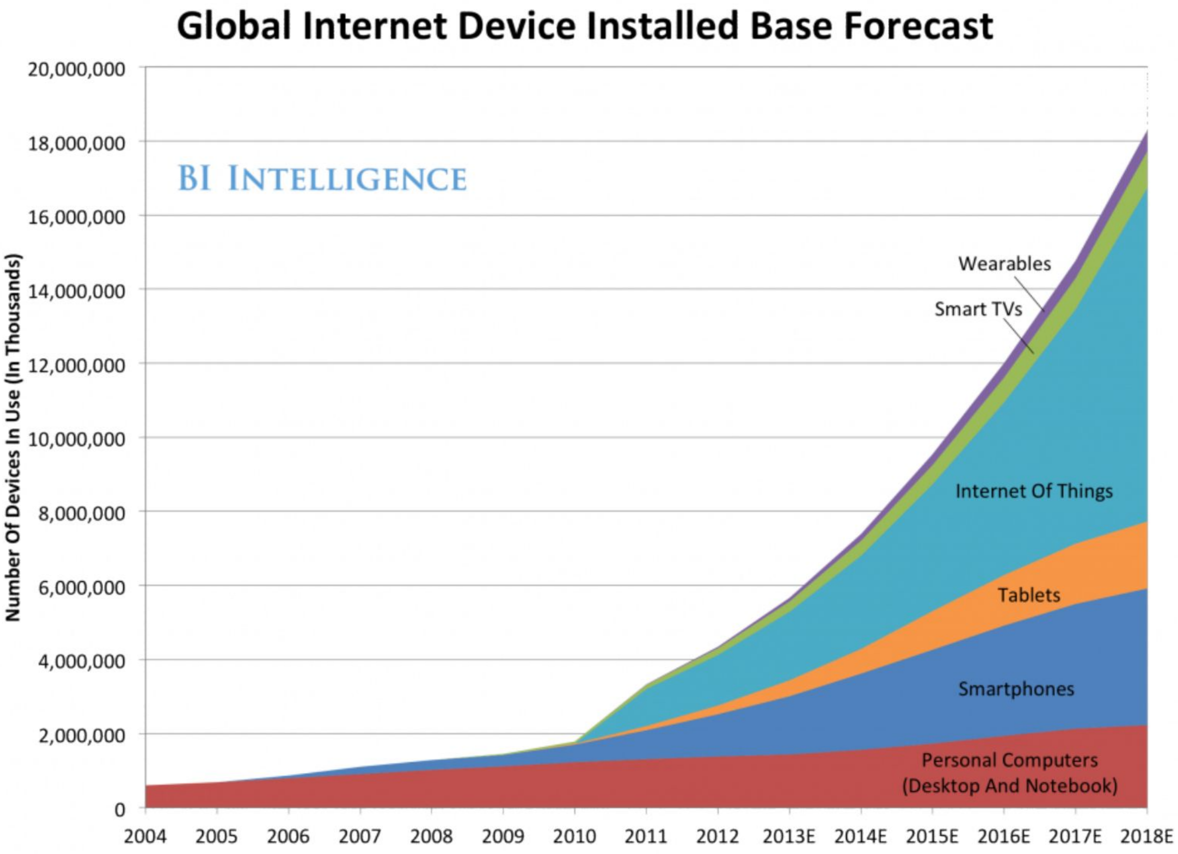
	State-of-the-art		VETPLC		
#	ALLSEQS	VETPLC-SEQS	VETPLC-TSEQS-2	VETPLC-TSEQS-5	VETPLC-TSEQS-10
1	N	N	Y	Y	Y
2	N	N	Y	Y	Y
3	N	N	Y	Y	Y
4	N	N	Y	Y	Y
5	N	N	Y	Y	Y
6	N	N	Y	Y	Y
7	N	N	Y	Y	Y
8	N	N	Y	Y	Y
9	N	N	Y	Y	Y
10	N	N	Y	Y	Y

Firmalice: Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware

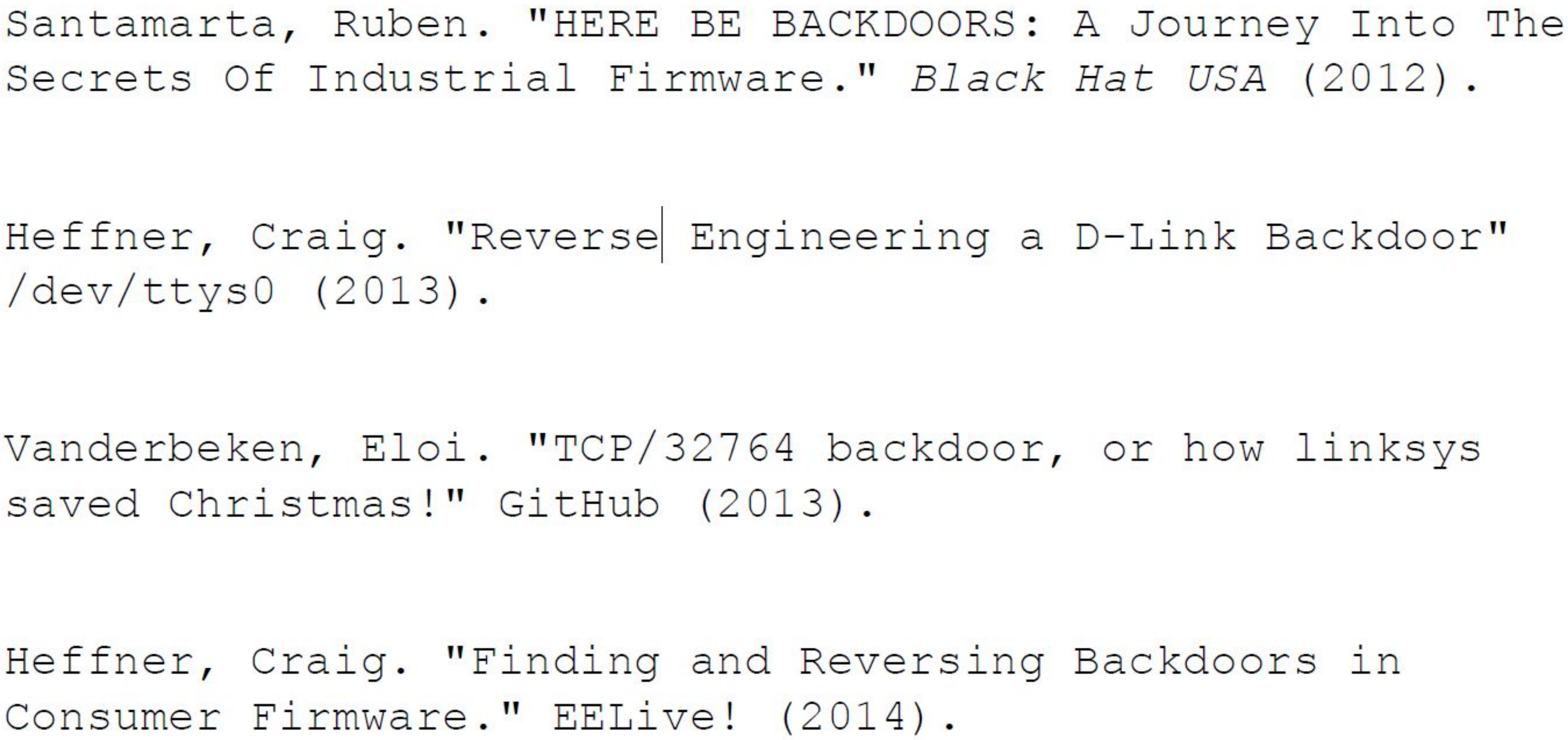
Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, Giovanni Vigna

NDSS 2015

The Rise of Firmware



Emergence of Backdoors



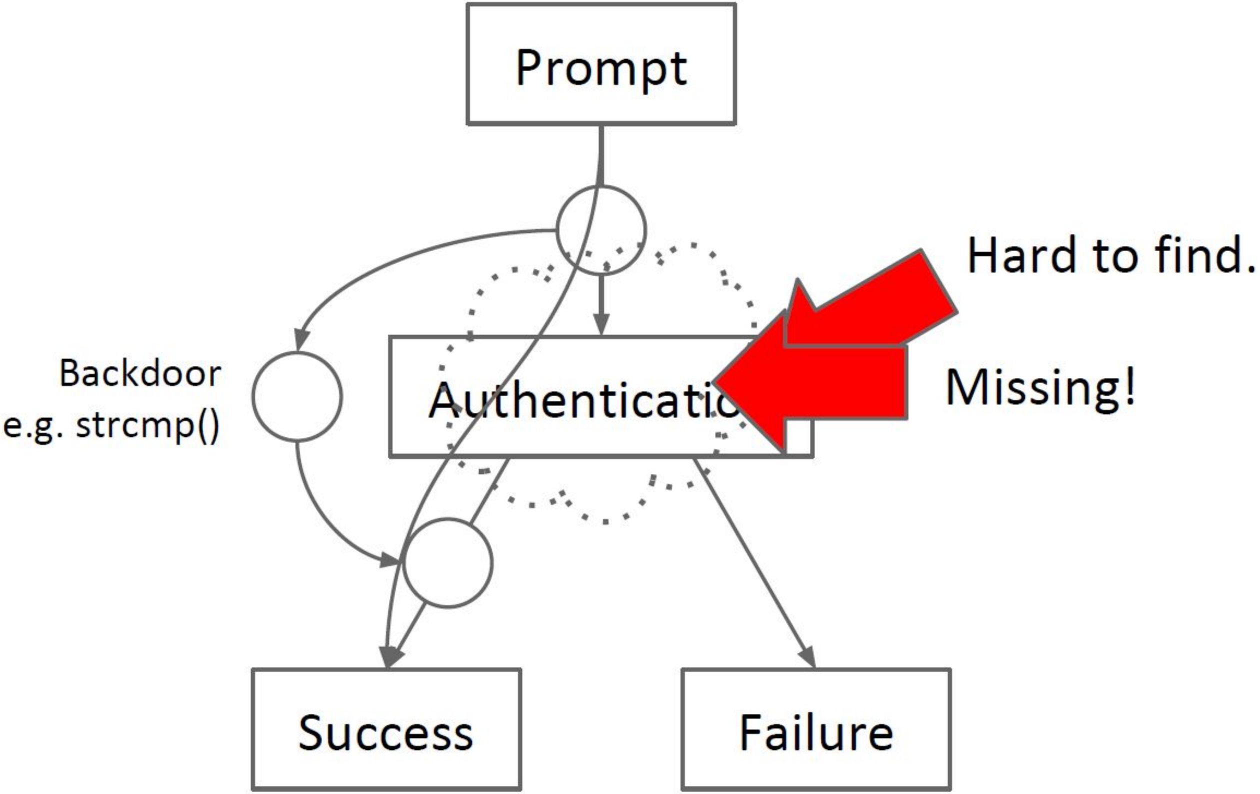
Santamarta, Ruben. "HERE BE BACKDOORS: A Journey Into The Secrets Of Industrial Firmware." *Black Hat USA* (2012).

Heffner, Craig. "Reverse Engineering a D-Link Backdoor" /dev/ttys0 (2013).

Vanderbeken, Eloi. "TCP/32764 backdoor, or how linksys saved Christmas!" GitHub (2013).

Heffner, Craig. "Finding and Reversing Backdoors in Consumer Firmware." EELive! (2014).

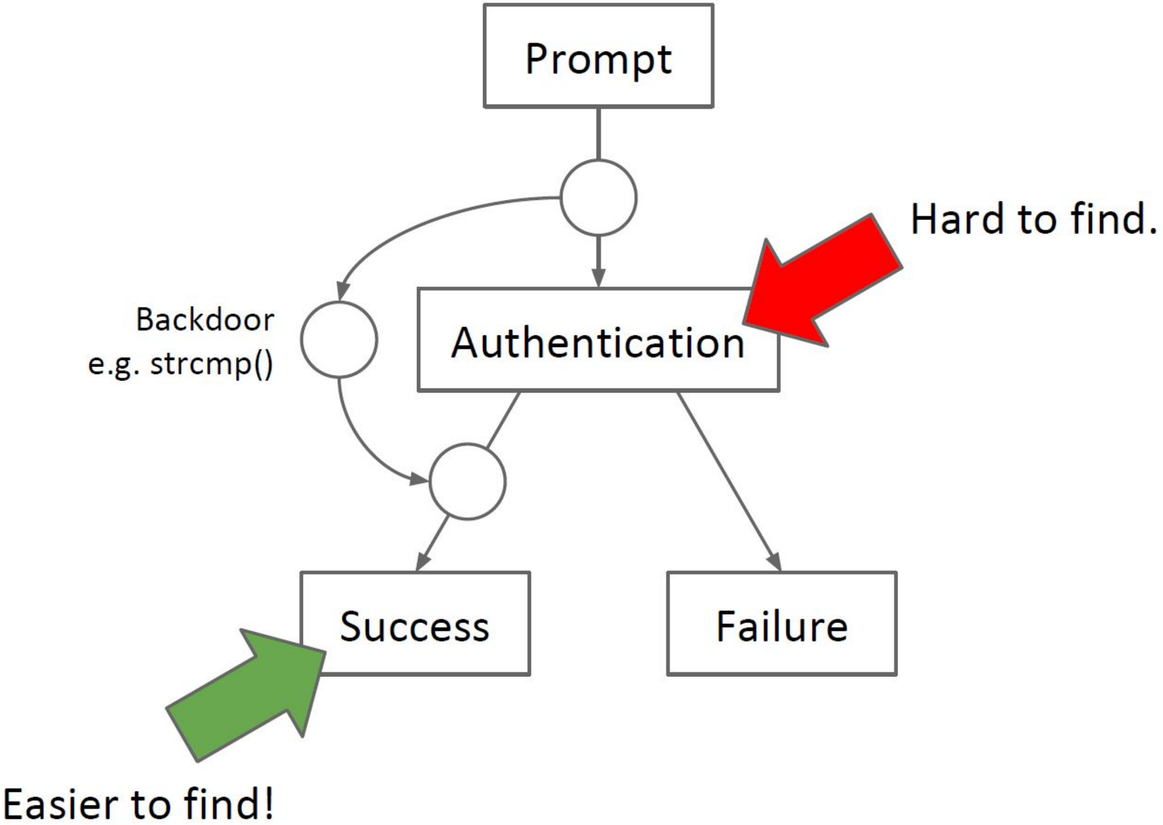
Backdoor Discovery



Hard to find.

Missing!

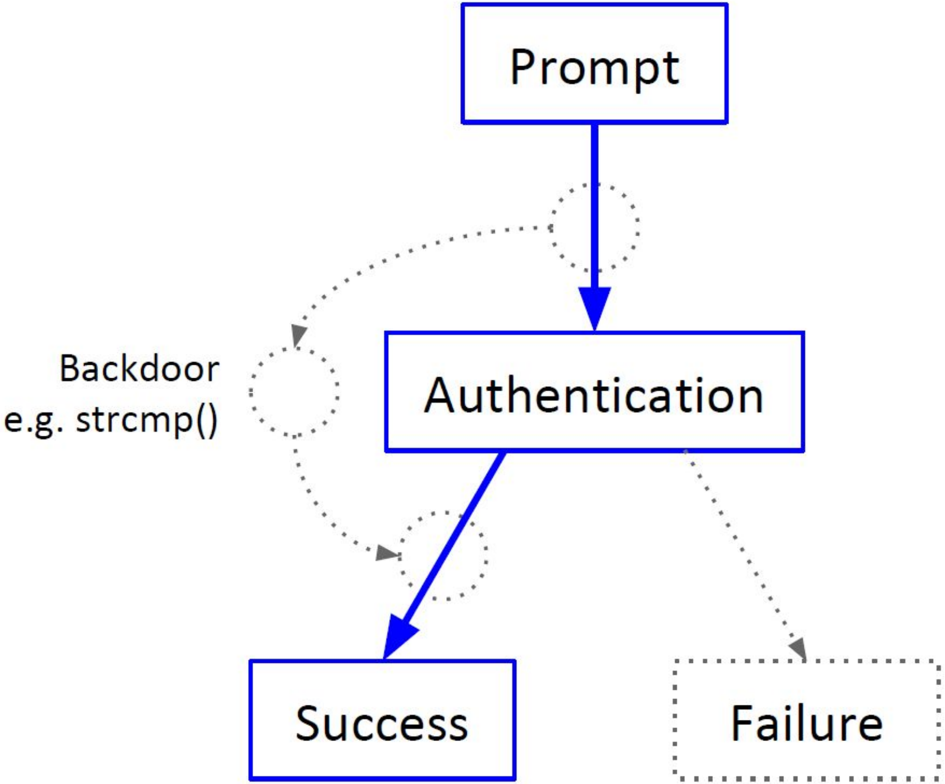
Solution: Input Determinism



Hard to find.

Easier to find!

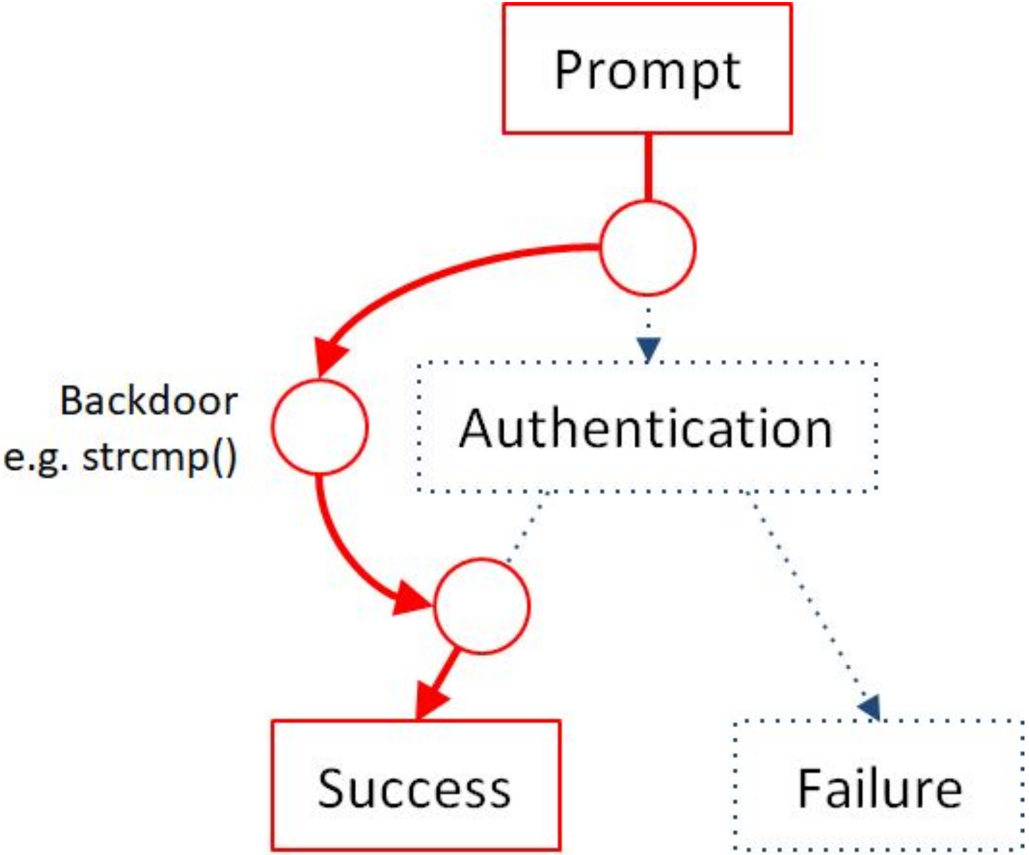
Input Determinism



Can we determine the input needed to reach the success function, just by analyzing the code?

The answer is NO

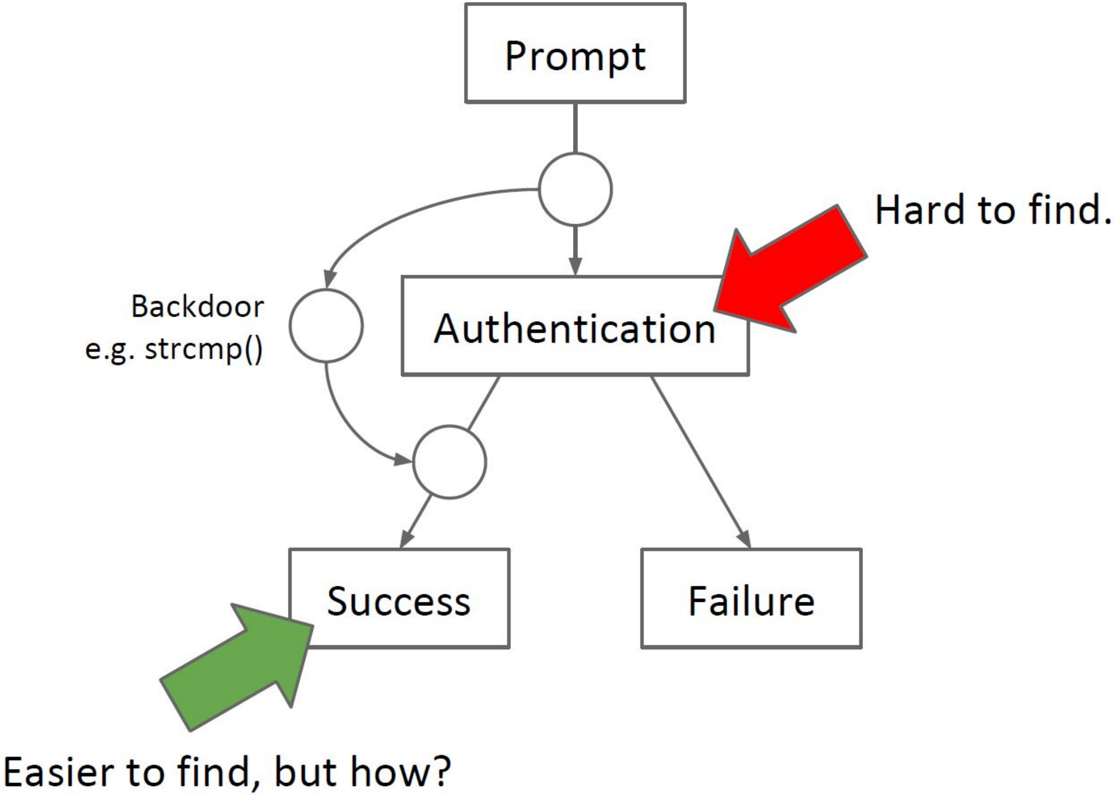
Input Determinism



Can we determine the input needed to reach the success function, just by analyzing the code?

The answer is YES

Challenge

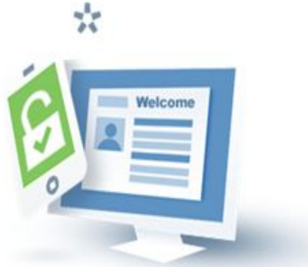


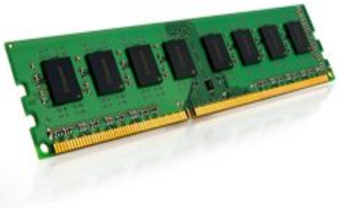
Hard to find.

Easier to find, but how?

Finding “Authenticated Point”

- Without OS/ABI information:





With ABI information:

Firmalice

Inputs:

➔ Firmware Sample

➔ Security Policy

Challenges:

➔ Large binary programs

➔ Unrelated user input

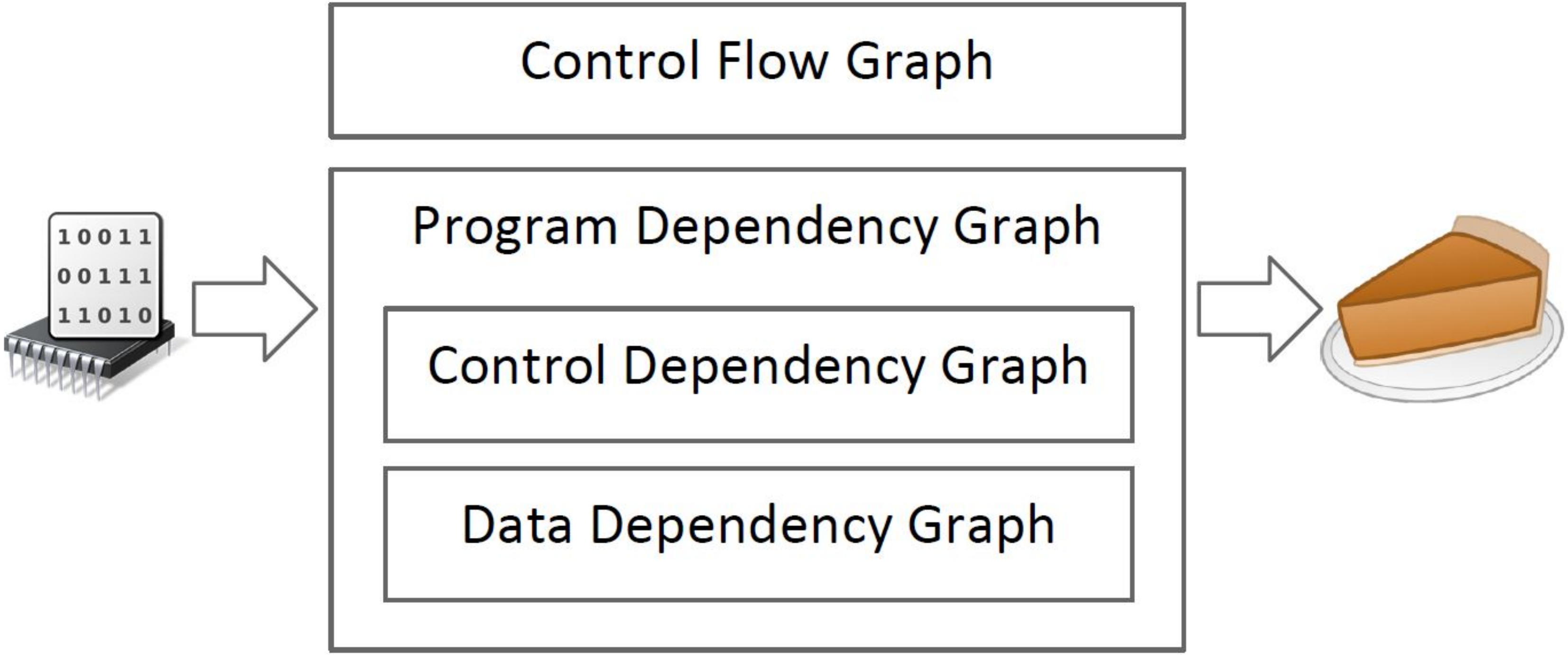
Analysis Steps:

➔ Static Analysis (backwards program slicing)

➔ Dynamic Symbolic Execution

➔ Authentication Bypass Check

Static Analysis



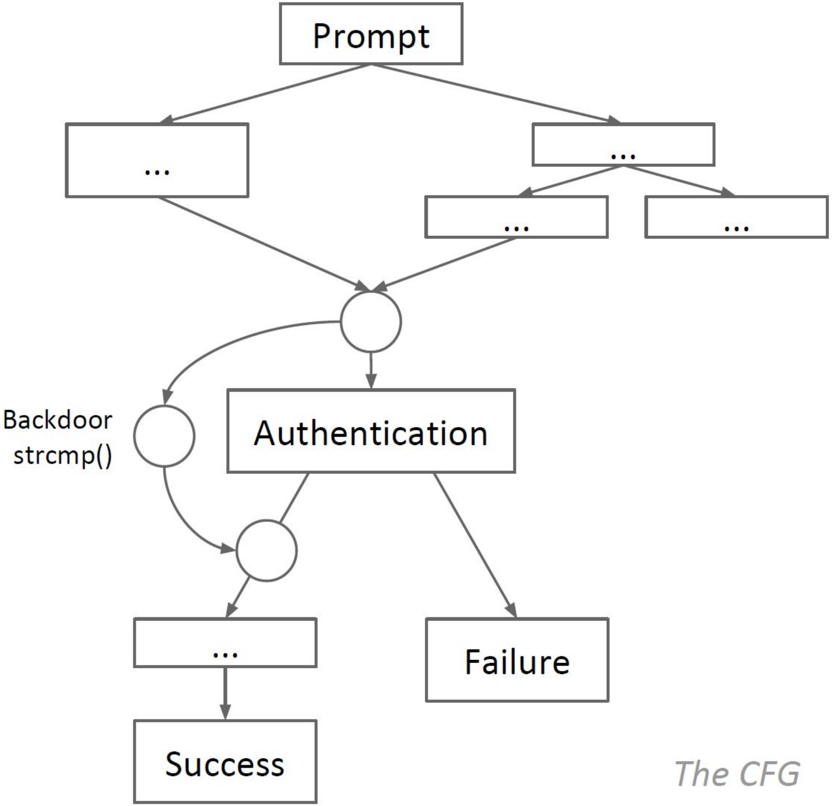
Control Flow Graph

Program Dependency Graph

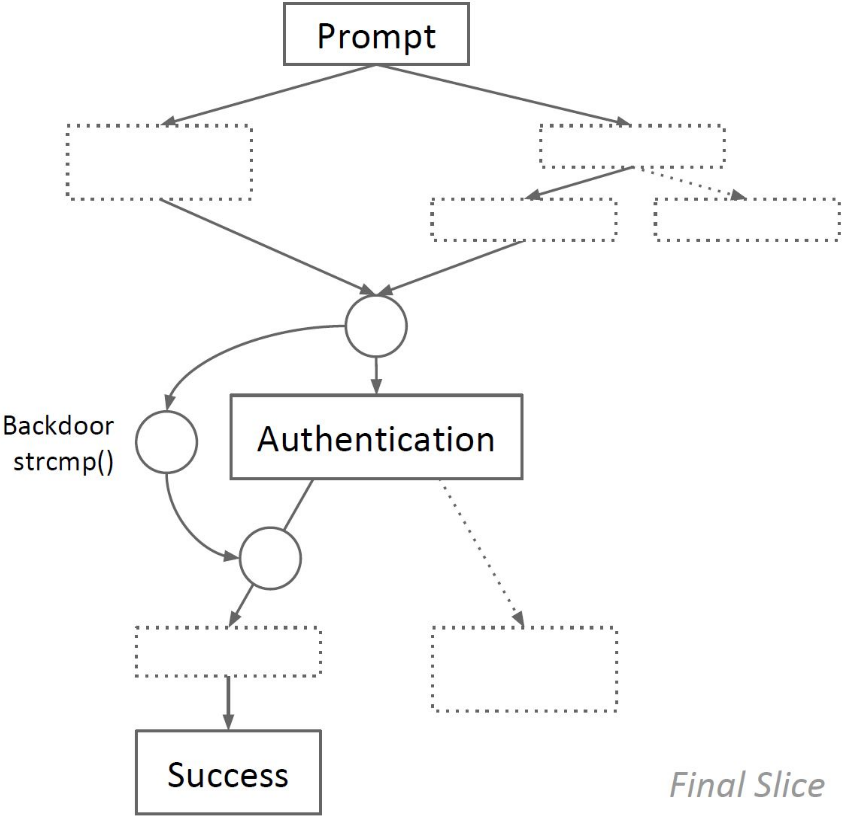
Control Dependency Graph

Data Dependency Graph

CFG

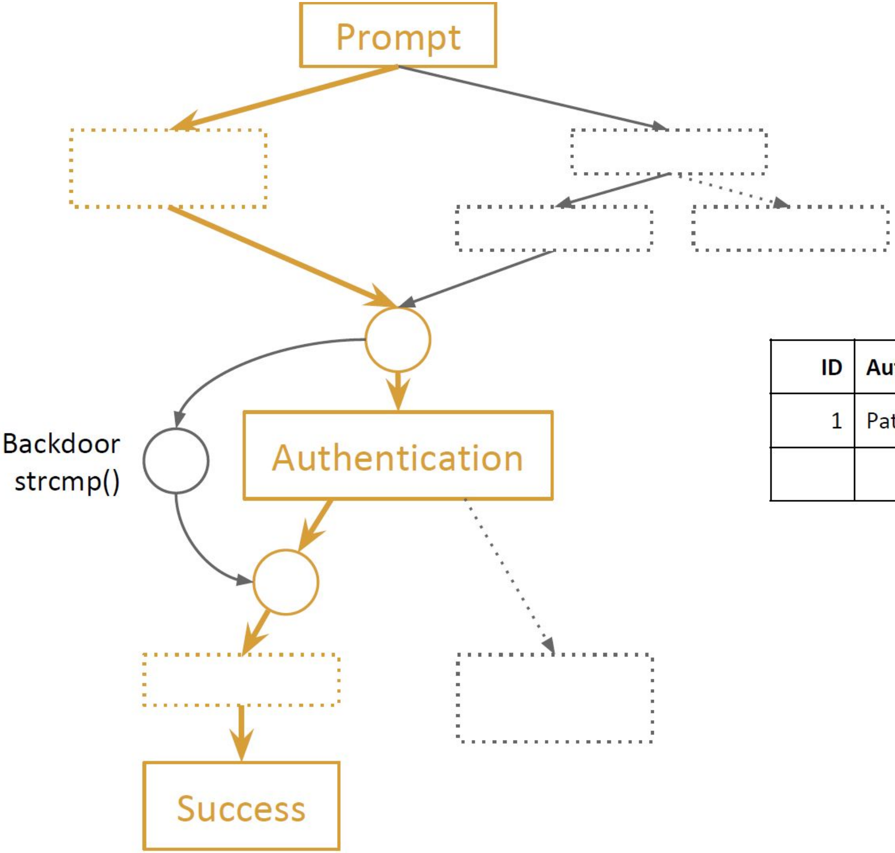


The CFG



Final Slice

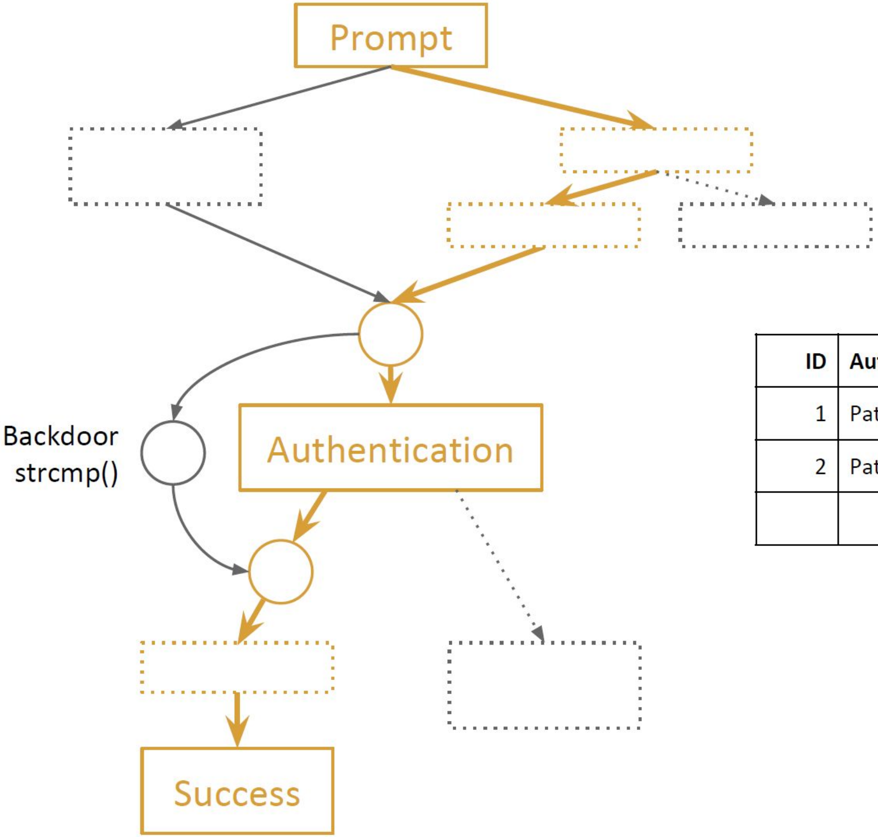
Dynamic Symbolic Execution



ID	Authenticated Paths
1	Path 1

Path 1

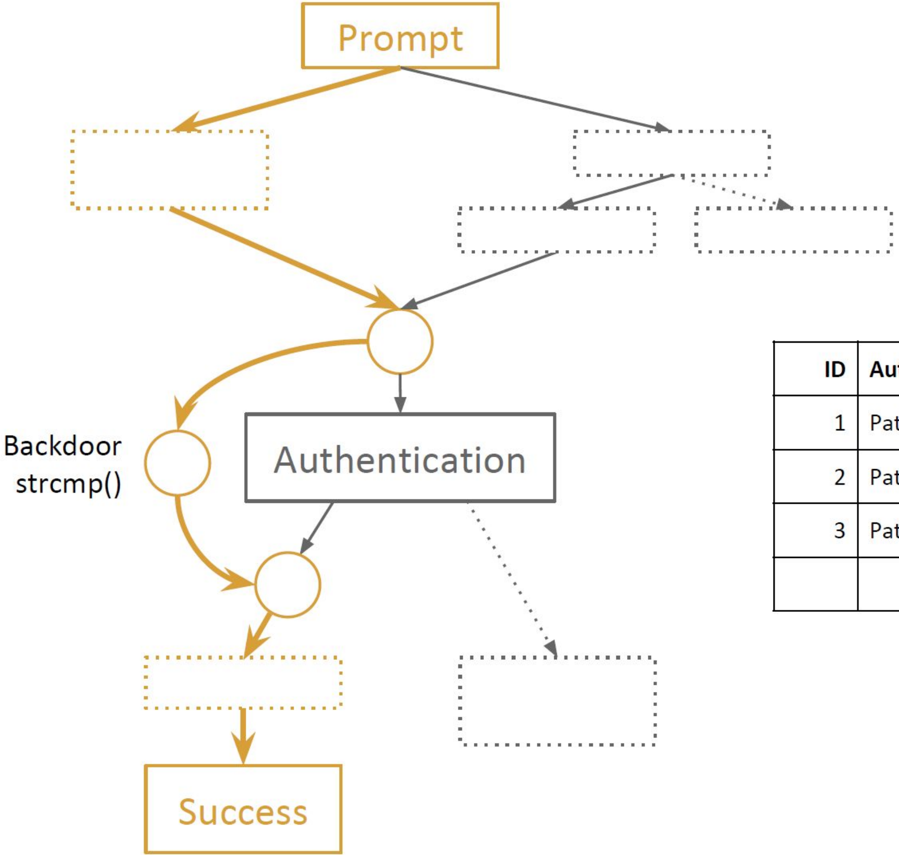
Dynamic Symbolic Execution



ID	Authenticated Paths
1	Path 1
2	Path 2

Path 2

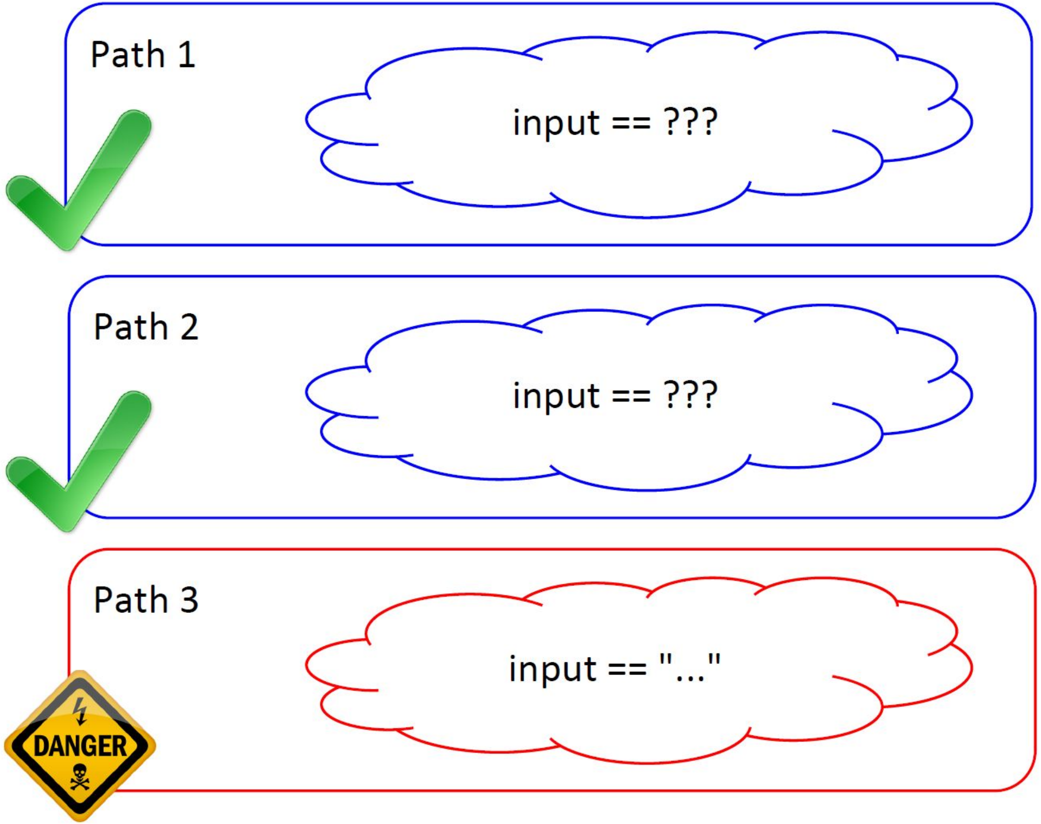
Dynamic Symbolic Execution



ID	Authenticated Paths
1	Path 1
2	Path 2
3	Path 3

Path 3

Authentication Bypass



Path 1

input == ???

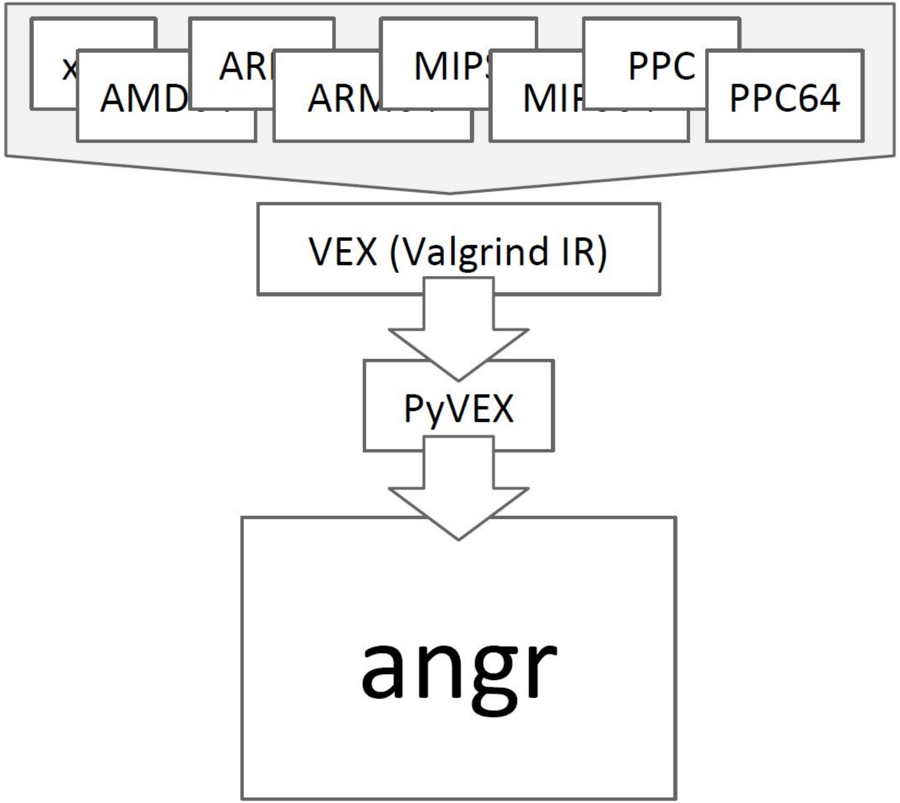
Path 2

input == ???

Path 3

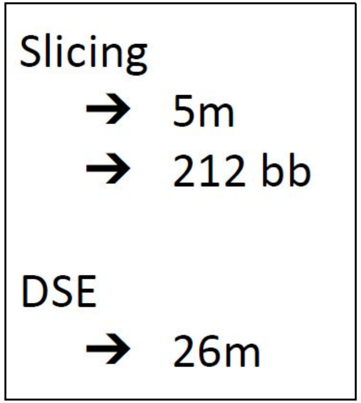
input == "..."

Implementation



Backdoor Example: 3S Vision N5072

- Linux embedded device.
- HTTP server for management and video monitoring.
- Security Policy
  - Authentication required for footage access
  - "Image-Type" header
- Backdoor
  - Hard-coded user credentials
  - Username: 3sadmin
  - Password: 27988303



Slicing

➔ 5m

➔ 212 bb

DSE

➔ 26m