Northeastern University

Attacks on Industrial Control Logic and Formal Verification-Based Defenses

Ruimin Sun, *Alejandro Mera, Long Lu, David Choffnes*

Northeastern University
r.sun@notheastern.edu

Motivation

2010

Stuxnet disrupts Iranian nuclear program

2013

Australia's largest satellite company

New York dam floodgates compromised

2014

Hospital drug infusion pumps hacked

Michigan traffic light hacked

2015

German steel mill furnace destroyed

Car transmission and brakes controlled

Ukraine power grid knocked offline

2017

Merck pharma global production shutdown by ransomware ($1B loss)

Maersk Shipping global shutdown by ransomware ($250M loss)

2018

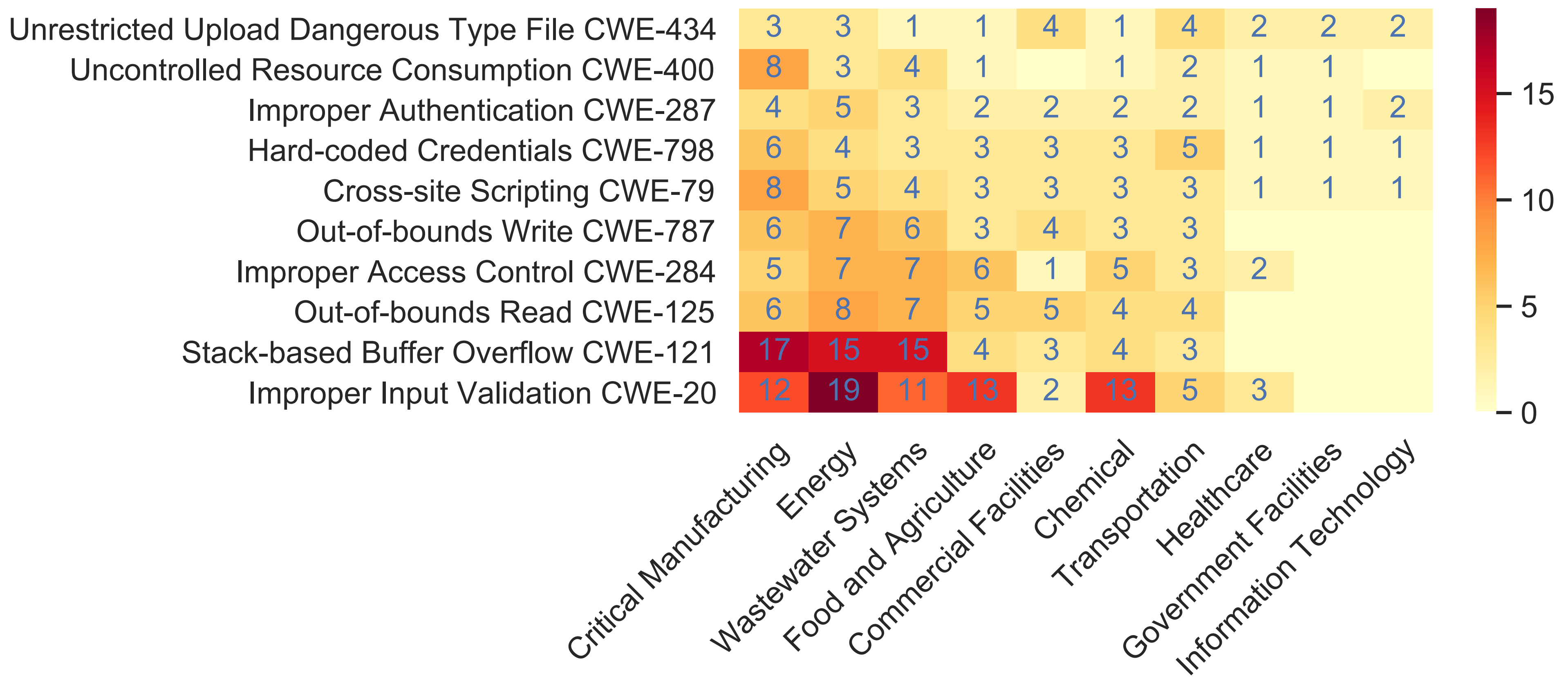
Trisis/Triton: Malware designed to compromise safety

2019

Global Aluminum producer shutdown by ransomware

2021

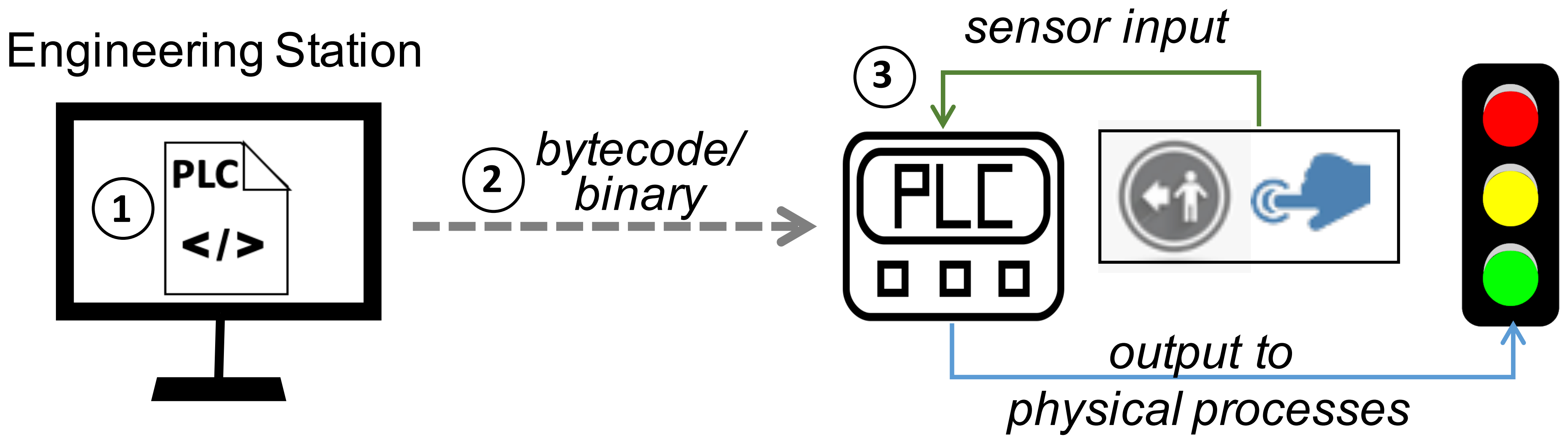
Motivation



	Critical Manufacturing	Energy	Wastewater Systems	Food and Agriculture	Commercial Facilities	Chemical	Transportation	Healthcare	Government Facilities	Information Technology
Unrestricted Upload Dangerous Type File CWE-434	3	3	1	1	4	1	4	2	2	2
Uncontrolled Resource Consumption CWE-400	8	3	4	1		1	2	1	1	
Improper Authentication CWE-287	4	5	3	2	2	2	2	1	1	2
Hard-coded Credentials CWE-798	6	4	3	3	3	3	5	1	1	1
Cross-site Scripting CWE-79	8	5	4	3	3	3	3	1	1	1
Out-of-bounds Write CWE-787	6	7	6	3	4	3	3			
Improper Access Control CWE-284	5	7	7	6	1	5	3	2		
Out-of-bounds Read CWE-125	6	8	7	5	5	4	4			
Stack-based Buffer Overflow CWE-121	17	15	15	4	3	4	3			
Improper Input Validation CWE-20	12	19	11	13	2	13	5	3		

Figure 1: The reported common weaknesses and the affected industrial sectors. The notation denotes the number of CVEs.

Background of PLC



Attacks and Defenses

- Control Logic Modification
  - attacks that can change the behavior of PLC control logic.
    - *program payload/code modification*
    - *program input manipulation*.
- Formal Verification
  - unique and practical to the PLC industry.
  - PLCs have limited memory and are less tolerant to false positives
    - the controlled physical processes are safety-critical.
  - used in the industry
- Existing research tend to be ad-hoc

A Motivating Example

```
TYPE Light : (Green, Yellow, Red); END_TYPE;
PROGRAM TrafficLight
  VAR_INPUT
    SensorNS : BOOL; SensorEW : BOOL;
    ButtonNS : BOOL; ButtonEW : BOOL;
  END_VAR

  VAR_OUTPUT
    LightNS : Light := Green;
    LightEW : Light := Red;
  END_VAR

  IF LightNS = RED AND LightEW = RED AND NOT(ButtonNS)
        AND NOT(SensorEW) THEN
    (* turn green when light is red, button is reset,
         and no emergency detected *)
    LightNS := Green;
  ELSIF LightNS = GREEN AND LightEW = RED AND SensorEW
        THEN
    (* light must change when emergency approaches in
         EW direction *)
    LightNS := Yellow;
  ELSIF LightNS = GREEN THEN
    LightNS := Green;
  ELSE THEN
    LightNS := Red;
  END_IF;

  (* The EW light status changes in a similar way *)
  (* Omitted *)
END_PROGRAM
```

if emergency vehicles are approaching

whether pedestrians press the button to cross the intersection

the status of lights at the NS and the EW directions. By default, the NS direction is green, and the EW direction is red.

the logic of changing light status based on the sensor inputs.

A simplified traffic light program written in Structure Text.

- controls the light
  - (e.g. green, yellow, red)
  - directions
    - north-south (NS)
    - east-west (EW)

A Motivating Example

```
TYPE Light : (Green, Yellow, Red); END_TYPE;
PROGRAM TrafficLight
  VAR_INPUT
    SensorNS : BOOL; SensorEW : BOOL;
    ButtonNS : BOOL; ButtonEW : BOOL;
  END_VAR

  VAR_OUTPUT
    LightNS : Light := Green;
    LightEW : Light := Red;
  END_VAR

  IF LightNS = RED AND LightEW = RED AND NOT(ButtonNS)
        AND NOT(SensorEW) THEN
    (* turn green when light is red, button is reset,
        and no emergency detected *)
    LightNS := Green;
  ELSIF LightNS = GREEN AND LightEW = RED AND SensorEW
        THEN
    (* light must change when emergency approaches in
        EW direction *)
    LightNS := Yellow;
  ELSIF LightNS = GREEN THEN
    LightNS := Green;
  ELSE THEN
    LightNS := Red;
  END_IF;

  (* The EW light status changes in a similar way *)
  (* Omitted *)
END_PROGRAM
```

- **Normally**

when the NS light is red, and an emergency vehicle is sensed in the NS direction, the sensor will be ON *until the NS light is switched to green*.

- **Attacks**

1. switch the emergency sensor ON (e.g. SensorNS := TRUE) when the NS light is red and the EW light is green
2. switch it OFF (e.g. SensorNS := FALSE) when the NS light is red and the EW light is yellow

Green lights ON simultaneously for NS, EW

A Motivating Example

```
MODULE main
   IVAR
      button_NS: boolean;
      button_EW: boolean;
      sensor_NS: boolean;
      sensor_EW: boolean;
   VAR
      light_NS: {RED, YELLOW, GREEN};
      light_EW: {RED, YELLOW, GREEN};
   ASSIGN
      init(light_NS) := GREEN;
      init(light_EW) := RED;

      next(light_NS) := case
         light_NS = RED & light_EW = RED & button_NS =
              FALSE & sensor_EW = FALSE: GREEN;
         light_NS = GREEN & light_EW = RED & sensor_EW
              = TRUE: YELLOW;
         light_NS = GREEN: GREEN;
         TRUE: {RED};
      esac;

      next(light_EW) := case
         light_EW = RED & light_NS = RED & button_EW =
              FALSE & sensor_NS= FALSE: GREEN;
         light_EW = GREEN & light_NS = RED & sensor_NS
              = TRUE: YELLOW;
         light_EW = GREEN: GREEN;
         TRUE: {RED};
      esac;

   SPEC AG ! (light_NS = GREEN & light_EW = GREEN)
```

- Formal Verification
  - model the program using SMV
  - NuSMV
    - verify the property
    - obtain the counterexamples

Specifies the property to verify
e.g. the green lights of NS and EW
can never be ON simultaneously

A Motivating Example

```
MODULE main
   IVAR
      button_NS: boolean;
      button_EW: boolean;
      sensor_NS: boolean;
      sensor_EW: boolean;
   VAR
      light_NS: {RED, YELLOW, GREEN};
      light_EW: {RED, YELLOW, GREEN};
   ASSIGN
      init(light_NS) := GREEN;
      init(light_EW) := RED;

      next(light_NS) := case
         light_NS = RED & light_EW = RED & button_NS =
               FALSE & sensor_EW = FALSE: GREEN;
         light_NS = GREEN & light_EW = RED & sensor_EW
               = TRUE: YELLOW;
         light_NS = GREEN: GREEN;
         TRUE: {RED};
      esac;

      next(light_EW) := case
         light_EW = RED & light_NS = RED & button_EW =
               FALSE & sensor_NS= FALSE: GREEN;
         light_EW = GREEN & light_NS = RED & sensor_NS
               = TRUE: YELLOW;
         light_EW = GREEN: GREEN;
         TRUE: {RED};
      esac;

   SPEC AG ! (light_NS = GREEN & light_EW = GREEN)
```

- Formal Verification

  NuSMV
  - verify the property
  - obtain the counterexamples

```
-> State: 1.1 <-
  light_NS = GREEN
  light_EW = RED
-> Input: 1.2 <-
  button_NS = FALSE
  button_EW = FALSE
  sensor_NS = FALSE
  sensor_EW = TRUE
-> State: 1.2 <-
  light_NS = YELLOW
-> Input: 1.3 <-
  sensor_EW = FALSE
-> State: 1.3 <-
  light_NS = RED
-> Input: 1.4 <-
-> State: 1.4 <-
  light_NS = GREEN
  light_EW = GREEN
```

Systematization Methodology

- Threat model
  - assumptions on source code, bytecode/binary, runtime.
- Security goal
  - Confidentiality, Integrity, and Availability
- Weakness
  - the flaws triggered to perform the attacks.
- Detection to evade
  - the detection that fails to capture the attacks
- Challenges *in **defending** the attacks*
  - the advance of attacks, and the insufficiency of defenses.
- Defense focus
  - the specific research topic in formal verification
  - e.g. behavior modeling, state reduction, specification generation, and verification.

Attack papers

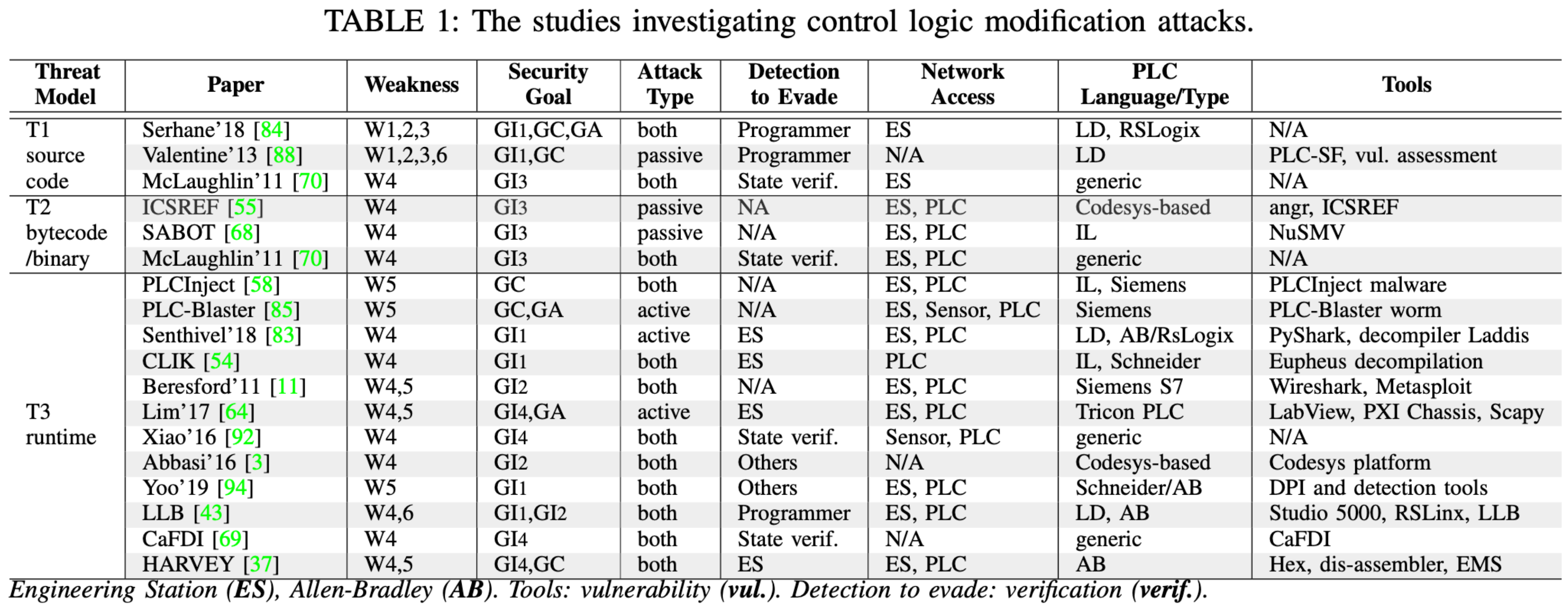


TABLE 1: The studies investigating control logic modification attacks.

Threat Model	Paper	Weakness	Security Goal	Attack Type	Detection to Evade	Network Access	PLC Language/Type	Tools
T1 source code	Serhane'18 [84]	W1,2,3	GI1,GC,GA	both	Programmer	ES	LD, RSLogix	N/A
	Valentine'13 [88]	W1,2,3,6	GI1,GC	passive	Programmer	N/A	LD	PLC-SF, vul. assessment
	McLaughlin'11 [70]	W4	GI3	both	State verif.	ES	generic	N/A
T2 bytecode /binary	ICSREF [55]	W4	GI3	passive	NA	ES, PLC	Codesys-based	angr, ICSREF
	SABOT [68]	W4	GI3	passive	N/A	ES, PLC	IL	NuSMV
	McLaughlin'11 [70]	W4	GI3	both	State verif.	ES, PLC	generic	N/A
T3 runtime	PLCInject [58]	W5	GC	both	N/A	ES, PLC	IL, Siemens	PLCInject malware
	PLC-Blaster [85]	W5	GC,GA	active	N/A	ES, Sensor, PLC	Siemens	PLC-Blaster worm
	Senthivel'18 [83]	W4	GI1	active	ES	ES, PLC	LD, AB/RsLogix	PyShark, decompiler Laddis
	CLIK [54]	W4	GI1	both	ES	PLC	IL, Schneider	Eupheus decompilation
	Beresford'11 [11]	W4,5	GI2	both	N/A	ES, PLC	Siemens S7	Wireshark, Metasploit
	Lim'17 [64]	W4,5	GI4,GA	active	ES	ES, PLC	Tricon PLC	LabView, PXI Chassis, Scapy
	Xiao'16 [92]	W4	GI4	both	State verif.	Sensor, PLC	generic	N/A
	Abbasi'16 [3]	W4	GI2	both	Others	N/A	Codesys-based	Codesys platform
	Yoo'19 [94]	W5	GI1	both	Others	ES, PLC	Schneider/AB	DPI and detection tools
	LLB [43]	W4,6	GI1,GI2	both	Programmer	ES, PLC	LD, AB	Studio 5000, RSLinx, LLB
	CaFDI [69]	W4	GI4	both	State verif.	N/A	generic	CaFDI
	HARVEY [37]	W4,5	GI4,GC	both	ES	ES, PLC	AB	Hex, dis-assembler, EMS

*Engineering Station (**ES**), Allen-Bradley (**AB**). Tools: vulnerability (**vul.**). Detection to evade: verification (**verif.**).*

Formal Verification papers (a truncated list)

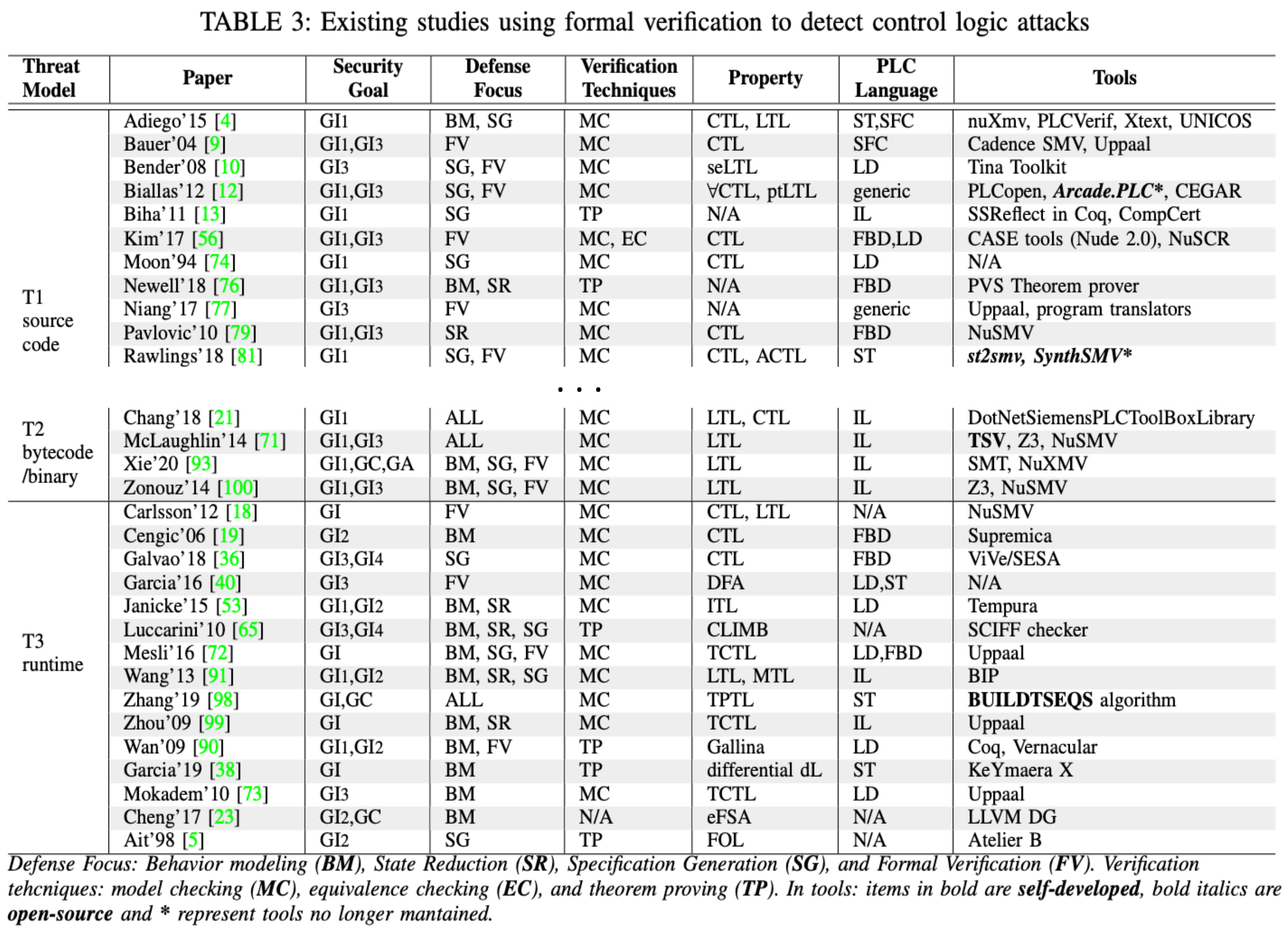


TABLE 3: Existing studies using formal verification to detect control logic attacks

Threat Model	Paper	Security Goal	Defense Focus	Verification Techniques	Property	PLC Language	Tools
T1 source code	Adiego'15 [4]	GI1	BM, SG	MC	CTL, LTL	ST,SFC	nuXmv, PLCVerif, Xtext, UNICOS
	Bauer'04 [9]	GI1,GI3	FV	MC	CTL	SFC	Cadence SMV, Uppaal
	Bender'08 [10]	GI3	SG, FV	MC	seLTL	LD	Tina Toolkit
	Biallas'12 [12]	GI1,GI3	SG, FV	MC	∀CTL, ptLTL	generic	PLCopen, ***Arcade.PLC**** , CEGAR
	Biha'11 [13]	GI1	SG	TP	N/A	IL	SSReflect in Coq, CompCert
	Kim'17 [56]	GI1,GI3	FV	MC, EC	CTL	FBD,LD	CASE tools (Nude 2.0), NuSCR
	Moon'94 [74]	GI1	SG	MC	CTL	LD	N/A
	Newell'18 [76]	GI1,GI3	BM, SR	TP	N/A	FBD	PVS Theorem prover
	Niang'17 [77]	GI3	FV	MC	N/A	generic	Uppaal, program translators
	Pavlovic'10 [79]	GI1,GI3	SR	MC	CTL	FBD	NuSMV
	Rawlings'18 [81]	GI1	SG, FV	MC	CTL, ACTL	ST	***st2smv, SynthSMV****
				. . .			
T2 bytecode /binary	Chang'18 [21]	GI1	ALL	MC	LTL, CTL	IL	DotNetSiemensPLCToolBoxLibrary
	McLaughlin'14 [71]	GI1,GI3	ALL	MC	LTL	IL	**TSV**, Z3, NuSMV
	Xie'20 [93]	GI1,GC,GA	BM, SG, FV	MC	LTL	IL	SMT, NuXMV
	Zonouz'14 [100]	GI1,GI3	BM, SG, FV	MC	LTL	IL	Z3, NuSMV
T3 runtime	Carlsson'12 [18]	GI	FV	MC	CTL, LTL	N/A	NuSMV
	Cengic'06 [19]	GI2	BM	MC	CTL	FBD	Supremica
	Galvao'18 [36]	GI3,GI4	SG	MC	CTL	FBD	ViVe/SESA
	Garcia'16 [40]	GI3	FV	MC	DFA	LD,ST	N/A
	Janicke'15 [53]	GI1,GI2	BM, SR	MC	ITL	LD	Tempura
	Luccarini'10 [65]	GI3,GI4	BM, SR, SG	TP	CLIMB	N/A	SCIFF checker
	Mesli'16 [72]	GI	BM, SG, FV	MC	TCTL	LD,FBD	Uppaal
	Wang'13 [91]	GI1,GI2	BM, SR, SG	MC	LTL, MTL	IL	BIP
	Zhang'19 [98]	GI,GC	ALL	MC	TPTL	ST	**BUILDTSEQS** algorithm
	Zhou'09 [99]	GI	BM, SR	MC	TCTL	IL	Uppaal
	Wan'09 [90]	GI1,GI2	BM, FV	TP	Gallina	LD	Coq, Vernacular
	Garcia'19 [38]	GI	BM	TP	differential dL	ST	KeYmaera X
	Mokadem'10 [73]	GI3	BM	MC	TCTL	LD	Uppaal
	Cheng'17 [23]	GI2,GC	BM	N/A	eFSA	N/A	LLVM DG
	Ait'98 [5]	GI2	SG	TP	FOL	N/A	Atelier B

*Defense Focus: Behavior modeling (**BM**), State Reduction (**SR**), Specification Generation (**SG**), and Formal Verification (**FV**). Verification tehcniques: model checking (**MC**), equivalence checking (**EC**), and theorem proving (**TP**). In tools: items in bold are **self-developed**, bold italics are **open-source** and * represent tools no longer mantained.*

Challenges

Due to the advances of attacks

- Expanded attack input surfaces.
- Predefined hierarchical memory layout.
- Confidentiality and integrity of the program I/O.
- Stealthy attack detection.
- Implicit or incomplete specifications.

Challenges

Due to the insufficiency of defenses

Behavior Modeling

- *Lack of plant modeling.*
- *Lack of modeling evaluation.*
- *State explosion.*

State Reduction

- *Lack of "ground truth" for continuous behavior.*
- *Implicitness and stealthy attacks from reduction*

Specification Generation

- *Lack of specification-refined programming.*
- *Ad-hoc and unverified specification translations.*
- *automated domain-specific property generation.*
- *Specification with evolved system design.*

Verification

- *Lack of benchmarks for formal verification*
- *Open-source automated verification frameworks.*
- *High demand for runtime verification.*

Future Research Recommendations

Behavior Modeling

- **Plant Modeling**
  - formalize more accurate and complete program behaviors.
    - granularity and level of abstraction for the plant models and the properties
  - consider the avoidance of state explosion
    - conditions of the plant that can trigger property violations.
- **Input manipulation verification**
  - input manipulation is widely adopted by the attackers.
  - Orpheus prototype in a PLC setting
    - event consistency checking between the program and the plant model
    - instrumentation on the input and output variables

Future Research Recommendations

State Reduction

- **the relationship between the “unrelated” states and the original program.**
  - “unrelated” states are trimmed to avoid state explosion problems.
  - security validation of “unrelated” code
    - unnoticeable unsafe behaviors?
    - e.g. a stealthy logger to leak program critical information.
  - automatic program cleaning for the stealthy code.

Future Research Recommendations

Specification Generation

- **Domain-specific property definition.**
  - consider domain-specific properties as a *hybrid program* consisted of continuous plant models as well as discrete control algorithms.
  - aim at security verification
  - support arithmetic operations, multitask programs, in various domains.
- **Incremental specification generation.**
  - aim at the fast-evolving system design
  - a full chain of behaviors, and update in a dynamic spectrum.
  - behaviors from new interactions should be compatible with existing properties.

Future Research Recommendations

Verification

- **Real-time attack detection.**
  - Depend on the engineering station, which have been exposed to various vulnerabilities
  - Consider a dedicated security monitor, e.g. TSV
  - Meet the scan cycle requirement
- **Open-source tools and benchmarks.**
  - have led to adhoc studies without evaluations on models and verification techniques.
  - vendor-independent, industrial complexities, and a set of security metrics
- **Multitasks Verification.**
  - use one task to spy or spread malicious code to the other co-located tasks,
    - E.g. PLCInject, PLC-Blaster
  - verify task intervals and priorities at various granularities.

Questions