iFinger: Intrusion Detection in Industrial Control Systems via Register-Based Fingerprinting

Kai Yang, *Student Member, IEEE*, Qiang Li, Xiaodong Lin, *Fellow, IEEE*, Xin Chen, *Member, IEEE*, and Limin Sun

***Abstract*— Nowadays, the industrial control system (ICS) plays a vital role in critical infrastructures like the power grid. However, there is an increasing security concern that ICS devices are being vulnerable to malicious users/attackers, where any subtle changing or tampering attack would cause significant damage to industrial manufacturing. In this paper, we propose the *iFinger*, a novel detection approach designed to mitigate ICS attacks adapting to various industrial scenes. We take advantage of an important insight that industrial protocol packets include register status values that are used to reflect the physical characteristics of ICS controllers. The iFinger utilizes register states to generate ICS fingerprints to detect malicious attacks on industrial networks. Specifically, the boolean logic represents every register state sequence of the ICS controller, and the deterministic finite automaton (DFA) generates a device fingerprint. To discover the ICS attacks, we propose two detection approaches based on device fingerprints, including passive and active detection. We present a prototype of the iFinger and conduct real-world experiments to validate its performance. Results show that our approach achieves 97.1% F1 score in ICS device identification. Furthermore, we simulate two typical ICS attacks (replacement and code modification) to validate the effectiveness of our iFinger in industrial networks. Our device fingerprints would detect those malicious attacks within 2s latency at 98.0% recall.**

***Index Terms*— Industrial control system (ICS), fingerprinting, intrusion detection.**

I. Introduction

CYBER-PHYSICAL systems (CPS) intertwine software components and physical processes, which are pervasive in diverse industries [1], such as manufacturing, automotive, energy, and medical monitoring. The control component plays a critical role in CPS, connecting cyberspace with physical worlds, e.g., Programmable Logic Controller (PLC), Remote Terminal Unit (RTU), and Distributed Control System (DCS). Compared to conventional computing systems, the control devices bring in a new security concern: being more vulnerable to attacks and being tightly coupled with physical processes. Any subtle attack on industrial controller devices would cause significant damage to the critical infrastructure [2]. For instance, an attacker uses a fake device to replace a real device [3], or remotely manipulates sensor and actuator data in the control component (e.g., "PLC blaster" [4]). However, the state-of-the-art of Intrusion Detection Systems (IDS) [5]–[8] are not capable of discovering those subtle changes and tampering attacks.

Security professionals seek practical approaches for monitoring and protecting industrial control systems (ICS) to ensure that they are operating safely and as intended. Prior works [9]–[12] proposed to utilize signals from sensors and actuator to discover malicious attacks to ICS devices. The problems are that (1) subtle attacks (replacement and code modification) cannot be detected by the sensor-based detection approach, (2) diverse and complicated manufacturing components would decrease the performance of the detection approach, e.g., control switch and circuit breaker. In this work, our goal is to build a detection model to mitigate subtle attacks to ICS devices in various industrial scenes.

Through an extensive investigation of ICS, we have two practical observations in manufacturing with ICS networks. First, an ICS control device has many registers in the memory, and their states are stable with time. For instance, a simple control device has two registers $\{(X0, Y0)\}$ with the state representation $\{(0,0),(1,1)\}$. When the physical characteristic of an ICS device keeps stable and invariant, its register states might also be fixed along time. In a nutshell, a subtle attack might have been being conducted if its registers change. Second, ICS devices typically run a variety of industrial protocols in the application layer, and register states might be encapsulated in the header of packets. For example, S7 protocol [13] packet has device register value at byte offset 19 in the header as function code "0×04". In our research, we have investigated 5 popular ICS protocols (S7 [13], FINS [14], CIP [15], Modbus [16], and SRTP [17]), which are widely used in industrial manufacturing. The manufacturers using these protocols are with more than 76% of market shares, according to the IPCS 2018 report [18]. Those ICS protocols have specific fields to store device register values in the runtime (detailed in Table II). In this paper, we leverage the

Manuscript received July 16, 2019; revised January 14, 2020; accepted February 26, 2020. Date of publication March 16, 2020; date of current version May 7, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC1201102, in part by the National Natural Science Foundation of China under Grant 61972024 and Grant U1636120, in part by the Key Program of National Natural Science Foundation of China under Grant U1766215, and in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDC02020500. *(Corresponding authors: Qiang Li; Limin Sun.)*

Kai Yang and Limin Sun are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100864, China, and also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: sunlimin@iie.ac.cn).

Qiang Li is with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China (e-mail: liqiang@bjtu.edu.cn).

Xiaodong Lin is with the School of Computer Science, University of Guelph, Guelph, ON N1G 1Y4, Canada.

Xin Chen is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100864, China.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2020.2980921

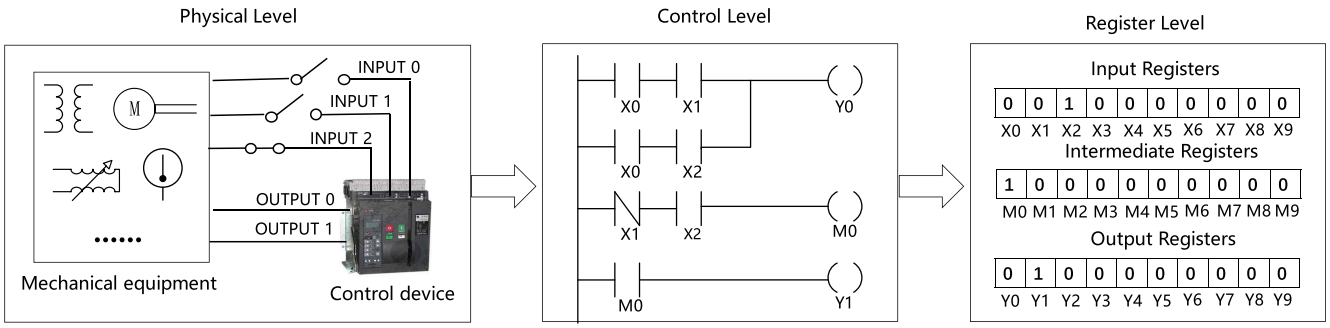


Fig. 1. An example of an ICS device from the physical level, control level to register level.

register-based characteristics to generate ICS device fingerprints to detect subtle attacks.

Although register states are capable of representing the physical characteristics of industrial devices, ICS attack detection has two challenges in practice. First, registers in ICS control devices might have a large number of states because of diverse and complicated control flows. For instance, when a state is associated with 100 registers, the space size will achieve 2^{100}, leading to the state explosion. Second, accessible register states are dependent on specific manufacturing networks. Although industrial protocols have packets consisting of register states, many registers might not be accessible or visible in ICS network traffic. In other words, some registers are visible in one ICS scene, but not visible in a different ICS scene.

To address these challenges, we propose *iFinger*, a novel detection approach designed to identify subtle ICS attacks, which applies to various industrial scenarios. Specifically, iFinger utilizes the boolean matrix to present the logic circuit of an ICS device and leverages deterministic finite automaton (DFA) to generate register state sequences for an ICS device. The node in DFA is a register state sequence, and the edge presents the transition relation from one node to another. For every ICS device, the iFinger will generate the DFA as its fingerprint. To discover ICS attacks, we propose two detection manners based on the DFA generated by the iFinger, including passive and active detection. For visible registers, passive detection will gather register values of ICS devices from the network trace. For invisible registers, the active detection will send crafted packets to ICS devices and gather corresponding register values. We detect ICS attacks based on a heuristic rule: if a register state sequence of a device is different from the current node in its DFA, the device is compromised or intentionally replaced by attackers.

To validate the effectiveness of the iFinger, we have implemented a prototype of the fingerprinting system and conducted real-world experiments. We evaluate 10 typical ICS devices from 5 top manufacturers, including Siemens, GE, Schneider, Omron, and Rockwell. Note that those devices cover 5 popular ICS protocols. The iFinger utilizes 10 DFA as device fingerprints for those ICS devices. Results show that our approach achieves 97.1% F1 score in ICS device identification. Furthermore, we simulate two typical ICS attacks (replacement and code modification) to validate the effectiveness of our iFinger in industrial networks. Our device fingerprints could detect those malicious attacks within 2s latency at 98.0% recall.

In brief, we make the following three contributions:

- We proposed the *iFinger*, an effective fingerprint generation approach for an ICS device based on its register states.
- The iFinger is the first work that leverages invisible and visible registers to detect subtle attacks in ICS manufacturing networks.
- We have conducted real-world experiments to validate the effectiveness of the iFinger, and the results show that it achieves 97.1% F1 score in device identification and 98.0% recall rate in attack detection.

The remainder of this paper is organized as follows. Section 2 provides the background of ICS controllers and register states. Section 3 presents the iFinger for generating ICS device fingerprints. Section 4 describes passive and active detection for discovering subtle ICS attacks. Section 5 presents the experimental evaluation of our iFinger. Section 6 surveys related work, and finally, Section 7 concludes our work.

II. ICS Controller and Register

In this section, we present the background of ICS controllers and register states.

A. ICS Controller

ICS control device (e.g., PLC and RTU) is a vital component that connects the network with manufacturing systems [19]. Typically, an ICS control device has a logic control circuit that is used to load the bytecode module. When an ICS device is running, its logic controller executes the bytecode in a fixed cycle, and periodically updates register states (e.g., "0" or "1") based on the bytecode. The register state sequence of the ICS device relies on the bytecode module and the logic control circuit. In a nutshell, different ICS devices would use different logic controllers and bytecode modules, and their register state sequences are capable of distinguishing from each other. Furthermore, any camouflage or subtle modification on ICS devices would dynamically change its register state sequence.

Figure 1 illustrates an example of an ICS device, including the physical manufacturing system, its logic control circuit,

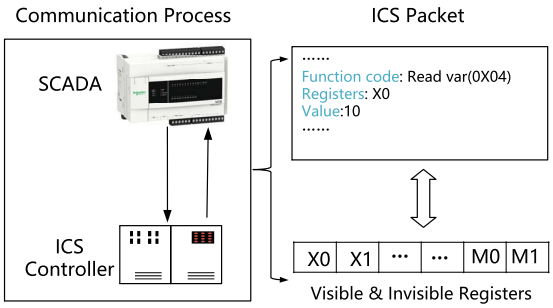


Fig. 2. An example of visible and invisible registers.

TABLE I
THE REGISTER TYPE

Register type	Illustration
Input register	store input signals
Output register	store onput signals
Intermediate register	store computation results between signals
Timer register	store timer value
Counter register	store count value
Holding register	store holding signal

and registers. In physical level, mechanical/manufacturing equipment is running as an actuator or sensor, and transmit physical signals to the logic controller via physical wires. Note that prior works [9]–[12] utilized the physical signal to detect ICS attacks. The logic controller receives input signals from physical equipment and stores those signals (input, output) in device registers. Here, the circuit in the logic controller consists of three inputs (X0, X1, X2), one intermediate (M0), and two outputs (Y0 and Y1). In register level, input registers store input signals; the output registers store output signals; intermediate registers store the computation result between other signals. When a signal value changes, its register state also changes immediately. Hence, the register state represents the signal from the physical manufacturing system. For example, the physical equipment sends the signal (0, 0, 1) to the controller and receives the output signal (0, 1) after the execution of bytecode in the logic controller. When the physical characteristic of an ICS device keeps invariant for a long period, its register state sequence is also fixed along time. In this paper, we leverage the register state sequence to generate device fingerprints for detecting subtle ICS attacks.

B. Register States

Aforementioned, when the bytecode of the ICS device is finalized, its register state sequence is invariant. At the running time, the register state has a specific value (e.g., "0" or "1") to represent the physical characteristics of ICS devices. For example, the input register of the voltage transformer has a "0" value because its liquid level is lower than the threshold. Figure 2 illustrates the communication process between an ICS device and the Supervisory Control And Data Acquisition (SCADA) system. For example, if an operator of a production task is monitoring a machine, he can finish this job by inspecting the register states. The SCADA system sends a request to the ICS device and receives its response with the register state value.

Different register types have different functions. Table I lists 6 register types, including input registers, output registers, intermediate registers, counter registers, timer registers, and holding registers. When manufacturing systems send physical signals, input registers store input signals, and the output registers store output signals. The intermediate register usually stores computation results between signals from other registers. The rest registers have complicated functions. The timer register stores a timer used to record the elapsed time for the SCADA system. The counter register stores a count value. When the counter is larger than the threshold, the counter register state is changed to "1". The holding register stores the holding signal controlled by the new signal from the external circuit.

Typically, register state values are encapsulated in the header of packets in industrial protocols. However, gathering register states, however, has a practical challenge in practice: many registers are not visible in network flows between SCADA and ICS controllers. The reason is that some registers are used to perform control tasks without being monitored by the SCADA system. Those registers are visible in network flows between SCADA and ICS controllers. Therefore, we divide registers into two categories: visible and invisible type. The visible register would be gathered from network flows, and the invisible register cannot be obtained from network flows. For example, the intermediate register "M0" is an invisible register in Figure 2 because the SCADA system uses input registers(X0, X1, and X2) to execute its industrial task. To address visible and invisible registers, we propose two detection manners, including passive and active detection. For visible registers, we directly collect them from network data-trace for detecting explicit attacks to ICS devices. For invisible registers, we send crafted packets to ICS devices to obtain them via parsing their response packets. Specifically, we have investigated popular ICS protocols and identified the fields offset in ICS packets. We use invisible registers to detect subtle attacks on ICS devices.

III. *iFinger*: ICS FINGERPRINT GENERATION

In this section, we present the *iFinger*, an approach designed to generate device fingerprints for automatically detecting ICS attacks. Figure 3 illustrates the overview of the iFinger, including the logic matrix (LM) module and register sequence generation (RSG) module. The inputs are a logic circuit and a register list of an ICS device, and the output is its fingerprint. Specifically, the LM module utilizes the boolean matrix to represent the logic circuit of the ICS control device. Given a logic circuit, we use the ladder logic programming language (LAD) to generate its coefficient boolean matrix. We extract all registers from the register list of the ICS device and use the boolean matrix to enumerate all register states. Then, the RSG module generates all possible sequences based on register states from the LM module. We use the

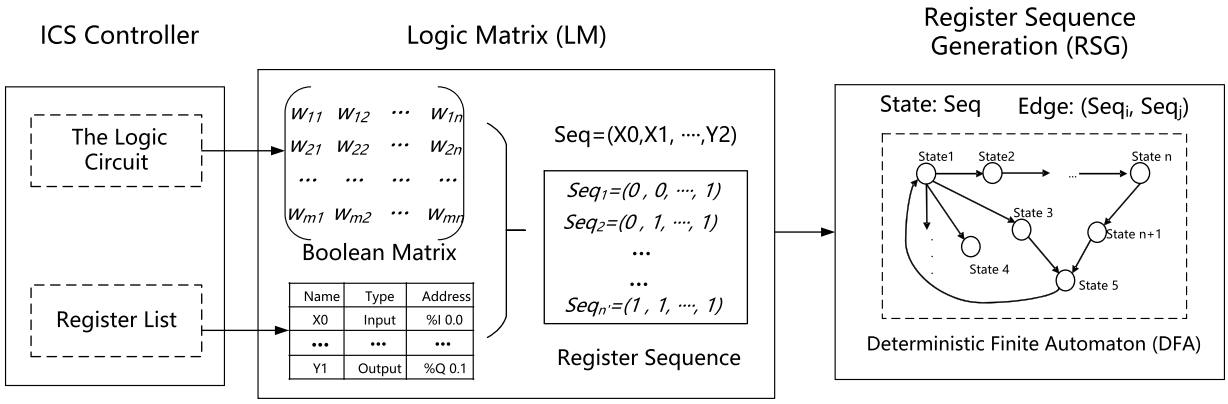


Fig. 3. The overview of the iFinger that generates industrial device fingerprints for detecting ICS attacks.

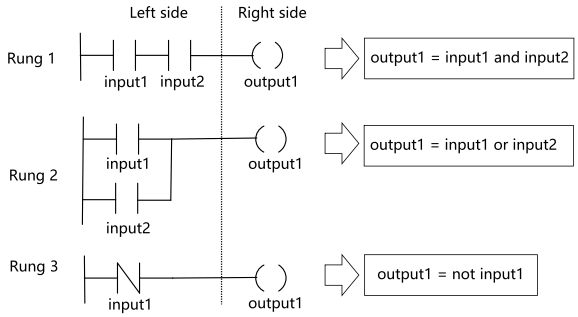


Fig. 4. An example of the logic circuit and the boolean matrix through LAD.

deterministic finite automaton (DFA) to present those register state sequences, where a node is a state sequence, and an edge indicates a transition between two state sequences. For an industrial device, we use the DFA to represent its fingerprint for detecting ICS attacks.

A. Logic Matrix

As we mentioned before, an ICS control device has a logic circuit and registers. The LM module will utilize the logic circuit and registers to enumerate all register states for an ICS device.

Typically, the standard IEC 61131-3 [20] claimed the programmable standard for industrial engineers. Here, we use the ladder logic programming language (LAD) to present a logic circuit in an ICS device. LAD has three boolean expressions, including "AND", "OR" and "NOT". Figure 4 depicts three boolean expressions in three rungs for LAD. Note that a rung is to present a row of boolean expressions in a boolean matrix. The symbol $|\ \ |$ and $|\backslash|$ on the left side of a rung present the input value of the register. The symbol $(\ \ )$ on the right side of a rung is the output value of the register. Given a controller circuit, we will use the LAD to convert it to a boolean matrix W, as follows:

$$W = \begin{vmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \dots & \dots & \dots & \dots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{vmatrix}$$

where every $w_{i,j}$ is a boolean value(0 or 1). Each row (rung) is to represent a boolean logic operation of register values, e.g., the logic circuit in Figure 4 is the boolean matrix $W = \{(0,0,0,0),(0,1,1,1),(1,1,0,0)\}$. The inputs are signals from physical equipment, and the outputs are boolean operation results of the logic circuit.

Aforementioned, an ICS control device has dozens of registers covering 6 typical types (input, output, intermediate, counter, timer, and holding registers). From a defensive perspective, we can obtain all registers from our ICS control devices. Every register has its name, address, and type, e.g., the register's name is "X0", its type is the input, and its address is "%I0.0". Note that the register store the value for corresponding signals in ICS devices, i.e., the input register for input signals, the intermediate register for intermediate signals, and the output register for output signals. Furthermore, the rest registers have complicated functions. For the timer register, its state is changed to "1" if the elapsed time is larger than the timer threshold. For the counter register, its state is changed to "1" if the counter is larger than the threshold. For the holding register, its state is changed to "1" if an extra signal from the outside circuit.

We use the boolean matrix to enumerate all register states for an ICS device. Note that input registers are depended on signals from physical equipment, and the boolean matrix can calculate other registers. For example, a boolean logic operation "Y0 = X0 and X1", "X0, X1" are input register states, and "Y0" is the calculation result as the output register state. We use the boolean matrices operation [21] to obtain other register states of the controller, as follows:

$$W \times \begin{vmatrix} Input \\ Registers \end{vmatrix} = \begin{vmatrix} Other \\ Registers \end{vmatrix} \quad (1)$$

Note that the matrix $|W|$ can be obtained through analyzing the bytecode module of the ICS controller. In each boolean logic operation, the boolean matrix enumerates all register states for generating ICS device fingerprints.

B. Register Sequence Generation

The RSG module would generate sequences based on register states extracted by the LM module. We utilize DFA to represent those register state sequences.

The algorithm 1 describes the DFA generation of the ICS device. The inputs are the boolean matrix M and the input

Algorithm 1 The DFA Generation for the ICS Device

Input: Matrix M, Input Register Set $REG = \{reg_1, \ldots, reg_n\}$
Output: DFA

1: **for** $\forall\ Seq \in$ Permutation(REG) **do**
2: order $+= 1$; $Set = \varnothing$;
3: **if** $\exists$ Holding, Timer, Counter $\in M$ **then**
4: Holding register state
5: timer $= 0$; counter $= 0$;
6: **end if**
7: **while** True **do**
8: **if** timer $\geq threshold_1$ **then**
9: Timer register state $= True$;
10: **end if**
11: **if** counter $\geq threshold_2$ **then**
12: Counter register state $= True$;
13: **end if**
14: $Seq' = M \times |Seq|$; *Equation 1*
15: **if** $Seq' \in Set$ **then** break;
16: **else** $Set = Set \bigcup Seq'$;
17: **end if**
18: Add a $node$ (Seq', order);
19: Add an edge ((Seq'_0, 0) $\rightarrow$ (Seq', order));
20: **for** $\forall$ DFA.$node_i$.order == cycle order **do**
21: Add an edge ($node_i \rightarrow$ (Seq', order));
22: Add the edge weight = timer;
23: **end for**
24: **end while**
25: **end for**

register set REG, and the output is the DFA. The DFA's node is a register state sequence, and its edge is a transition from one node to another. Line 1 utilizes input state sequences to generate other register sequences. For every input register, its states have two values, including "0" and "1". Given n input registers, we enumerate all sequences (2^n) based on their states. We use the variable ("$order$") to indicate the order of input-register state sequences, and the variable ("Set") to represent other-register state sequences that have been calculated (Line 2). The reason is that the SCADA system has an execution order to perform its industrial task. An ICS control device receives signals from the SCADA system and executes its logic circuit from the top rung to the bottom rung. Note that sequences generated by the matrix and the current register sequence share the same order in the DFA.

We use an example to illustrates the order of different sequences in the matrix M, as shown in Figure 5. The input register is "X0" with its space ("0" and "1"). The matrix M represents "Y1 equals M0" and "M0 equals X0". In the first cycle, the input register is "0" which is the first cycle in input state sequence space. Meanwhile, "Y1" and "M0" are both "0". In the second cycle, the input register turns to "1" and becomes the second cycle in the input state sequence. The "M0" is still "0" at the first rung, and turns to "1" at the second rung. The register "Y1" is still "0" in the second cycle. In the third cycle, "Y1" becomes "1" because "M0" is changed to "1".

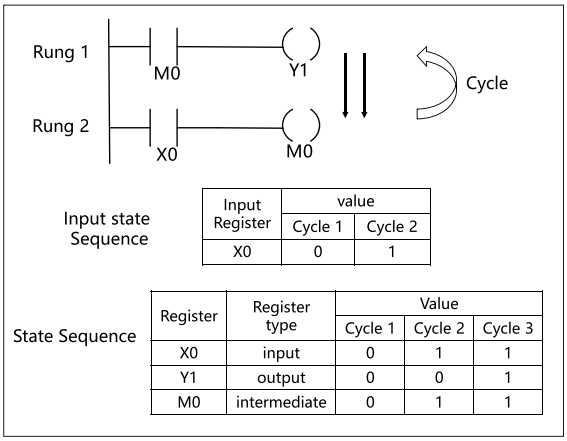


Fig. 5. An example of ICS control device execution order.

As we mentioned before, there are 3 complicated registers, including counter registers, timer registers, and holding registers. Those registers will intertwine input registers and the matrix M for generating other sequences. If M has those complicated registers, we use two variables ("timer" and "counter") to describe those registers' conditions. If those conditions are triggered, we change their state values for the M (Line 8-13). Line 14 is that the other-register sequence is calculated by the boolean matrix and the input-register state sequence according to the Equation 1. If this sequence does not appear, we add it into the Set; otherwise, we continue to the next input-sequence (Line 15-17). Then, we will use the sequence and its order as the node of the DFA in Line 18, denoting as (Seq', order). Line 19 shows that we add the edge from the first node to the new node. Every register sequence will be the starting point of the DFA matching process. If the order of the DFA node i is equal to the order of the new node, we also add the edge from i to it (Line 20-21). If the timer register exists, the timer value is the edge weight between two nodes. The reason is that the interval of the edge is to discover ICS attacks with a vast time offset. When we use the DFA to discover ICS attacks, the sequence matching transmits from the node i to the new node. The algorithm 1 outputs the DFA as the ICS device fingerprint. We use a heuristic rule: if a register state sequence of an ICS control device is different from its register-based model in DFA, it may be compromised or intentionally replaced by attackers.

IV. ICS Attack Detection

In this section, we first analyze 5 popular ICS protocols for obtaining register states from packets. Then, we describe two detection methods for discovering subtle ICS attacks, including passive and active detection.

A. ICS Protocol

ICS devices communicate with SCADA systems through a variety of industrial protocols in the application layer [22]. Packets in ICS protocols might have relevant information,

TABLE II
READING REGISTER STATES FROM PACKETS IN 5 POPULAR ICS PROTOCOLS

Protocol	request		response
	func (offset)	register_offset	value offset
S7	0X04(11)	22	19
Modbus	0X02(08)	09,10	10
Fins	0X0101(27,28)	30,31	30
Cip	0X0A(40)	32	44
Srtp	0X04(43)	44,45	44

such as device vendor, type, function, and register state. Table II shows 5 popular protocols that are widely adopted in industrial control systems, including S7 [13], FINS [14], CIP [15], Modbus [16], and SRTP [17]. In our research, we analyze the packet header from those 5 protocols, and find the field and offset for storing register state values. Below we detail those ICS protocols.

1) S7 Protocol: Siemens company proposed the S7 protocol, which is widely used in control devices, i.e., s7-300, s7-400, and s7-1200. Note that S7 is proprietary, and the Siemens Step 7 is the only official tool to communicate with ICS devices. We use the Tcpdump to capture the packets between "Siemens Step 7" and ICS devices. Then, we parse those packets to find how to extract register states from S7 devices. When a controller sends a packet (the function code is "0×04" at the offset 11) with registers address at the 22nd byte to an ICS device, it will respond a packet with the register state value (at the offset 19) to the controller.

2) Modbus Protocol: Modbus protocol is widely used by various ICS manufacturers, i.e., Schneider, Modicon, and Asea Brown Boveri (ABB). The Modbus protocol is an application layer protocol based on TCP/IP network stack. Typically, the tool "Unity Pro XL" is used to communicate the data with SCADA systems through the Modbus protocol. After analyzing their network flows, we find that the ICS device sends a packet with the register state value (at the offset 10) to the controller if it receives a packet (the function code is "0×02" at the offset 8).

3) FINS Protocol: Omron corporation proposed the factory interface network service (FINS) protocol for ICS devices. We use the tool "CX-Programmer" to communicate with ICS controllers and use Tcpdump to analyze their network flows. When a controller sends a packet (the function code is "0×0101" at the offset 27 and 28) to SCADA systems, we can obtain packets with the register state value (at the offset 30).

4) CIP: Common industrial protocol (CIP) is a unified communication architecture throughout the manufacturing industrials, including EtherNet/IP, DeviceNet, CompoNet, and ControlNet. We focus on the EtherNet/IP, which is widely used in a range of industries, including factory, hybrid, and process. The tool "RSLogix 5000" is an official software for the ICS controller to communicate with SCADA systems. We also use Tcpdump to read network flows for ICS devices. The request packet uses the function code "$0 \times 0a$" at the offset 40 in the packet, and the response packet contains the register state value at the offset 44.

5) SRTP Protocol: General Electric proposed service request transport protocol (SRTP) for transferring data between ICS controllers and SCADA systems. We use the "Proficy Machine Edition" to generate network flows for identifying the register field in packets. When a controller sends a packet (the function code is "0×04" at the offset 43) with registers address at the 44th byte and 45th byte to an ICS device, it responds a packet with the register state value (at the offset 44) to the controller.

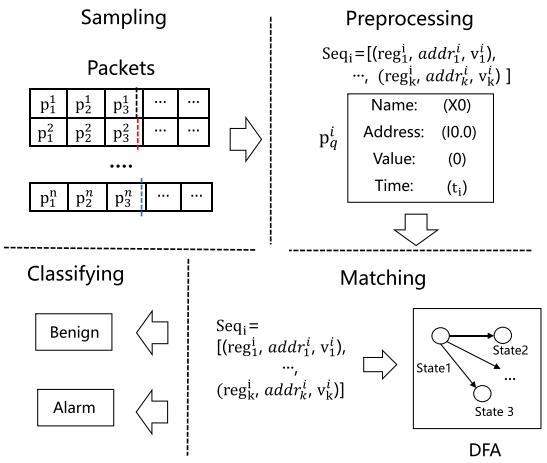


Fig. 6. Passive detection to ICS attacks.

B. Passive Detection

As we mentioned before, we directly collect visible register states in network flows to match with the DFA of the ICS device. Figure 6 depicts the overview of the passive detection for identifying ICS attacks. We start by collecting the ICS packets, followed by the preprocessing, which is used to extract register state values. Those register values are grouped into a state sequence in terms of the fixed period in network flows. Subsequently, the state sequence is used to match with the ICS fingerprint (DFA) generated by algorithm 1. If matched, we determine that the ICS device is running safely. Otherwise, we send an alarm to industrial administer because the ICS device may be compromised or intentionally replaced by attackers. Below we detail those four phases of passive detection.

1) Sampling: When a SCADA system executes an industrial task (e.g., controlling voltage transformer), it communicates with its ICS controller. Typically, packets with visible registers would be transmitted by the SCADA system in a fixed period. For example, an energy control automation system samples the registers of its controller per second. In the sampling phase, we keep packets with register values and filter out other packets. Then we identify the period of register occurrence in network flows, and present a circle of registers as a state sequence. Given the register packet set P, we divide those packets into a different group, which is presented as a register sequence. In the sampling phase, we obtain a sequence i

denoted as $Seq_i = \{p_1^i, p_2^i, \ldots, p_k^i\}$, where and i is the sequence identifier, and k is the order of packers.

2) Preprocessing: In the phase, we extract relevant information from ICS packets, including register value, register name, and register address. Note that those values are encapsulated in packets in ICS protocols listed by Table II. Specifically, we convert the packet p into the register tuple (reg, $addr$, v), where reg is the register name, $addr$ is register address, and v is the register value. Given the sequence i, it will be converted into the $Seq_i = \{(reg_1^i, addr_1^i, v_1^i), \ldots, (reg_k^i, addr_k^i, v_k^i)\}$. For every sequence, we also record its timestamp, denoted as (Seq_i, t_i). We use the sequence Seq_i as the input of the DFA to identify ICS attacks.

3) Matching: In the matching stage, we compare the sampled sequence Seq with the DFA of the ICS control device. Aforementioned, every node in DFA is a register sequence for benign ICS control devices, and the edge is the transition relation from one node to another. In the beginning, the initial $node_0$ will be matched and move to the next node in the DFA. We use the preserved node to present the last-matched node. In the matching stage, we iteratively match the following sequence with the next node of the preserved node according to its edge. If SCADA systems are running safely, the sequences will always successfully find a matched node in the DFA. Otherwise, we send an alarm about underlying risks in the SCADA system. Note that when the matrix M has the timer register, we need to add an extra matching condition. When the preserved node is updated, we also record its timestamp. We obtain the interval time between the timestamp of the current sequence t_i and the timestamp of the preserved node. We compare the interval time with the edge weight to identify a subtle attack. If the interval time is larger than twice of edge weight, we think the matching process to fail and send an alert to the SCADA systems.

C. Active Detection

Since many registers are invisible, we cannot obtain them from network flows. A well-designed attack will conceal themselves in the early stage and be too subtle to be aware of visible registers before they finally break out. For instance, the attacker modifies original code to malicious code, meanwhile retain primary functions, leading to unchanged visible registers in network flows. In manufacturing systems, any subtle change may cause server damages, such as a heater overheat. To detect subtle attacks, we propose active detection to obtain invisible registers for the DFA. Figure 7 depicts the overview of active detection. Like passive detection, active detection has the same other three modules, including the preprocessing, matching, and classifying components. The different place is that we utilize the active probing to obtain the registers, including visible and invisible ones. Here, we focus on the active phase, which is different from passive detection.

In the active probing phase, we craft the packets with particular fields (func code and register type) listed in Table II. For invisible registers, we send crafted packets to the SCADA system and parse their response packets. The problem, here, is that the SCADA system may be the time-critical infrastructure, and our active probing should not affect its regular operation. We use a heuristic rule to mitigate the inference of active probing: if the ICS network flow size is less than an upper bound, we send probing packets to obtain register states. We assume the upper bound is the B, which indicates the maximum of the network flow packets without any negative inference. The network flow count is the Y, and the number of active probing packets is the $B-Y$. We leverage the convolutional neural network (CNN) to predict the network flow count in the future.

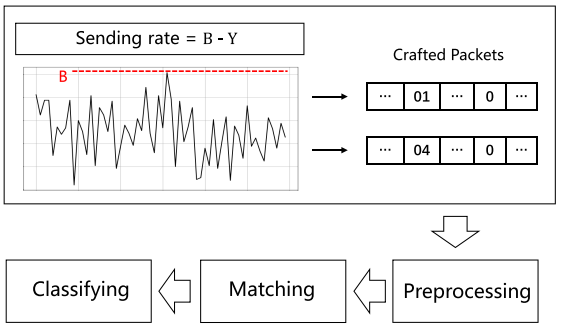


Fig. 7. Active detection to ICS attacks.

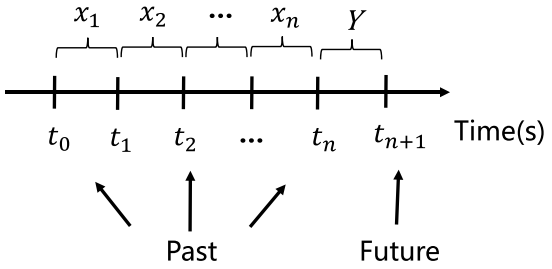


Fig. 8. The CNN model predicts the amount of network traffic in the future based on the past network flows.

The CNN generates the prediction model, as $f(X) \rightarrow Y$, where X is the past network flows, and Y is the amount of network traffic in the future. Figure 8 shows the input and output of the CNN prediction model. The past network flow is represented as $X = \{x_1, x_2, \ldots, x_n\}$. The x_1 is the traffic size for the previous second, and x_n the traffic size before n seconds, The output is the traffic size in the next second, denoted as the Y. The CNN learns the prediction model $y = f(x_1, x_2, \ldots, x_n)$.

In the first and second layers of CNN, we use Long Short-Term Memory (LSTM) to obtain the sequential relationship among the inputs ($x_i \in X$). LSTM is well-suited to solve sequence learning tasks based on time series data. The formula 2 is the following form,

$$\omega = \theta\left[\overrightarrow{LSTM}(x_1, x_2, \ldots, x_n)\right] \tag{2}$$

where the θ is the activation function. When the $\overrightarrow{LSTM}(x_i)$ is less than 0, the θ is assigned to 0. Otherwise, θ is assigned to the value of $\overrightarrow{LSTM}(x_i)$.

Then, we deploy two convolutional layers in the CNN model. A convolutional layer has many filters to its input, and obtain classification features. We use the convolutional formula as follows,

$$Q(k) = \sum_{i=0}^{n} h(k-i)R(i) \tag{3}$$

where the $h(k-i)$ is convolution kernel, $R(i)$ is convolution signal and n is the length of convolution signal. Further, we use the softmax function to calculate the probability of y. Given that the output y has k numbers, the softmax function normalizes y into a probability distribution, as follows,

$$p_k = \frac{exp(y_k)}{\sum_{i=0}^{n} exp(y_i)} \tag{4}$$

To obtain the network flow size in the future, we use the largest probability of y_k, where $Y = \{k| \max\ p_k\}$

To learn the prediction model based on CNN, we use the loss function (categorical cross entropy) to derive its parameters, as follows:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} 1_{y\in C_c} log\ p\,[y \in C_c] \tag{5}$$

where $\sum_{i=1}^{N}\sum_{c=1}^{C}$ is the observations of the training dataset, $i \in N$ and $c \in C$. The term $1_{y_i \in C_c}$ is the indicator function of the ith observation belonging to the cth category. The $p(y_i \in C_c)$ is the probability predicted by the model for the ith observation to belong to the cth category. Finally, we use the $(B-Y)$ as the number of active probing for register values. We obtain request/response packets pairs in the active probing period.

V. Evaluation

In this section, we first present the performance of ICS fingerprinting in real-world experiments. Then, we present the effectiveness of ICS fingerprinting based on two detection approaches: network flows and active probing. We finally detect two well-designed attacks in the ICS network, including code modification and device replacement.

A. Performance of ICS Fingerprinting

To validate the performance of ICS fingerprinting, we utilize 10 ICS devices to conduct real-world experiments. Figure 9 shows those 10 real-world ICS devices in our experiments, which consist of 5 popular industrial manufacturers, including Siemens, GE, Schneider, Omron, and Rockwell. We further list their usage and details in Table III. Typically, we adopt 3 Siemens devices (1200, 400, and 300) in the S7 protocol, which is used in traffic lights, liquid mixing, and elevator controller. For Schneider vendor, we use two devices (Quantum and Twido) in the Modbus protocol, which is used in cylinders and converyor. For Rockwell vendor, we use two devices (1756 and 1769) in CIP protocol, which is used in NC machine and liquid cooling. We adopt 2 Omron devices (CJ2M and CP1L) in FINS protocol, which is used in the Transformer and Trrible Machine. For GE vendor, we use the versamax device in the SRTP protocol, which is used in Cycle Car.

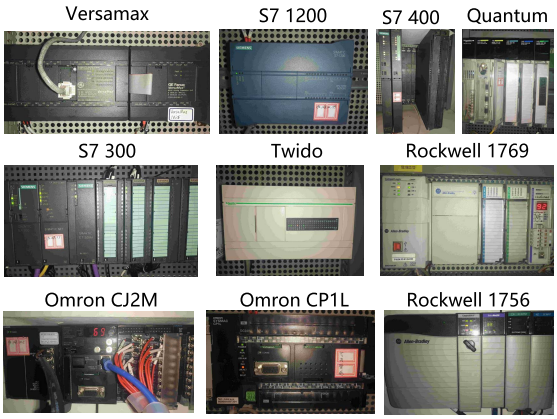


Fig. 9. 10 ICS devices in real-world experiments.

TABLE III
Details of 10 ICS Devices

Device (abbr.)	Use	Detail
GE Versamax (versa)	Cycle Car	Compact controller
Siemens s7 1200 (1200s7)	Traffic Light	Compact controller
Siemens s7 400 (400s7)	Liquid Mixing	Process controller
Siemens s7 300 (300s7)	Elevator	Universal controller
Schneider Quantum (quan)	Air Cylinder	Large controller
Schneider Twido (twido)	Conveyor	Base controller
Omron CJ2M (ocj2m)	Transfomer	Compatible controllers
Rockwell 1756 (r1756)	NC Machine	Control Logix and GuardLogix controllers
Rockwell 1769 (r1769)	Liquid Cooling	Compact Logix controller
Omron CP1L (ocp1l)	Trrible Machine	High performing controllers

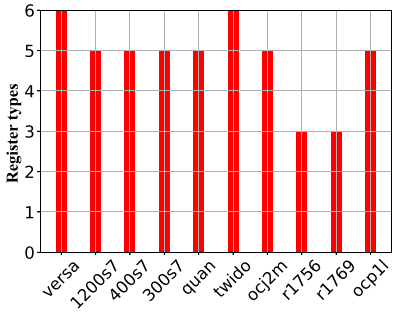


Fig. 10. The number of registers and the types from ICS develop documents.

Figure 10 depicts the number of registers and types among those 10 ICS devices. We found Schneider Twido has the least number of registers with 594, and with 6 register types. This controller is for small tasks and designed for affordable prices. For Siemens s7 1200, it has 65552 registers with 5 register

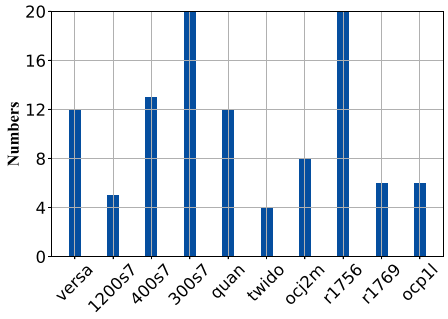


Fig. 11. The node count in DFA for ICS devices.

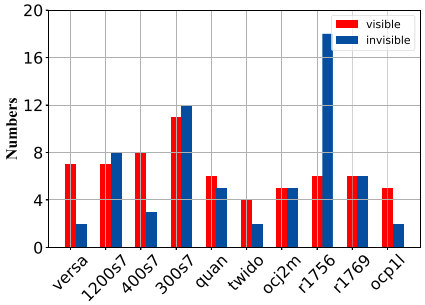


Fig. 12. Visible and invisible registers number in these controller.

types. Note that possible register sequences are enormous, as 2^n, where n is the number of registers. To reduce the register number of ICS devices, we only use the visible and invisible registers. Figure 12 illustrates the number of visible registers and invisible registers of these controllers which are in use. For Rockwell 1756, its invisible register is 3 times large than its visible registers. If we only monitor visible registers for intrusion detection, it might have a substantial probability miss the subtle attack target to invisible registers.

We present the registers from two aspects in the evaluation (Figure 10 and 12), where Figure 10 shows the registers number and the types, Figure 12 shows the number of visible and invisible register. There exist several types of registers in ICS devices, such as input register and output register. These registers can be divided into visible and invisible. If the register shows its value in SCADA data flow, it is divided into the visible register. Otherwise, it will be the invisible register. Based on those registers, we use the DFA to generate those ICS device fingerprints. Figure 11 illustrate the node count of DFA among 10 ICS devices. Compared with large register spaces (Figure 10), our node count in DFA is much smaller, only dozens of nodes. The Siemens s7-300 has 16960 registers, and its DFA fingerprint only contains 20 nodes, which is useful in real ICS intrusion detection.

B. Detection Performance

1) Passive Detection: We put those 10 ICS devices together under an industrial switch. Network flows pass through the switch, and we obtain the dataset through its mirror port. Totally, we gathered 34,259 ICS packets. Figure 13 presents the overview of the f1 score of the detection via network flows. F1-score is the harmonic mean of precision and recall,

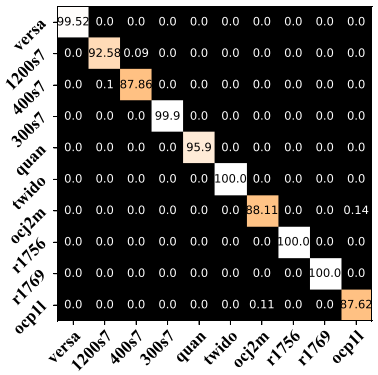


Fig. 13. F1 score of the detection via network flows.

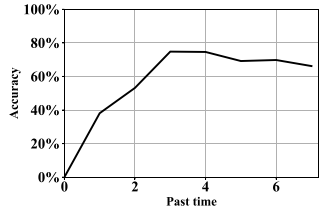


Fig. 14. The performance of CNN model along with the time length.

as follows,

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In this normalized confusion matrix, each row represents the instances in an actual category, while each column represents the instances in a predicted category. The diagonal of the matrix is the F1 score of each classification. We observe that the average f1 score has achieved at 96.09%. There is an error between device Ormon CP1L(row J, column G), Ormon CJ2M (row G, column J), and S7 1200 (row B, column C), S7 400 (row C, column B). The reason for the error is that these devices may share the same register state sequence between these devices, which means the registers state and their address are the same. The reason is that the limited visible registers increased the overlapped of different devices register states, from the same ICS protocol. In our experiment, Ormon CP1L and Ormon CJ2M were running on Fins protocol while S7 1200 and S7 400 were running on the S7 protocol.

2) Active Detection: As we mentioned before, many registers may be invisible in the network flow. We use the active probing to collect those invisible registers instead of monitoring network flows. We train the CNN prediction model for sending crafted packets. The CNN model is trained on an NVIDIA Tesla K80 graphics card. The graphics card is on a server with two Intel Xeon CPU E5-2650 v4 and 256GB 2400Mhz memory chips. We implement the CNN through Keras framework [23], where the dimension of the LSTM layer output is set as 50, and the dimension of the convolutional layer output is set as 50. The optimizer algorithm which we leverage in training is the *rmsprop* algorithm. The training epoch is set as 400, and in each epoch, the batch_size is set as 32.

We validate the length of the historical flow size for predicting the accurate flow size in the future. Figure 14 shows

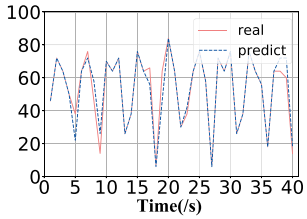


Fig. 15. The deviation between CNN prediction and real value.

TABLE IV
THE PERFORMANCE BETWEEN CNN AND SVM

Algorithm	*Datasize*	*Accuracy*
SVM	4000	67.3%
	3000	60.6%
	2000	51.0%
CNN	4000	74.8%

the curve of the accuracy along with the time length $|X|$. We observe that the CNN model achieves at 74.8% when it uses the last 3 seconds as the input. Although the performance CNN seems not promising, we claim that it is acceptable in practice. We use the CNN model to predict the network flow size in the 40 seconds. Figure 15 depicts the deviation between our CNN prediction and real results in this period. During this period, the sum number of the packets is 2128, and the CNN result is 2132. There is less than the 4-packet cumulative average deviation in total between the real and the prediction result. Although the CNN's performance is only 74.8%, the cumulative average deviation is close to several packets during the 40s. Hence, our CNN model can predict the network flow size for crafting packets of active probing.

Meanwhile, we compare the algorithm performance for the CNN method with a benchmark algorithm: SVM. The comparison result is showed in Table IV. We use the sklearn framework to implement the SVM classifier and set the parameter "gamma" as 1 and parameter "c" as 50. We range the training data from 2000 to 4000 to validate the relationship between system performance and the dataset size. The accuracy attains 51% with 3000 training data, improved to 60.6% with 3000 training data, and increase to 67.3% with 4000 training data. It can be seen that the SVM efficiency is weaker than CNN.

Furthermore, we validate the performance of active probing. Under the limited sending rate of ICS networks, we crafted packets and sent them to ICS devices. Figure 16 illustrates the performance of detection via active probing. We find that the average F1 score in active probing achieves at 99.1%, compared with the 96.09% of detection via network flows. These two detection methods achieve 97.1% F1 scores on average. The error of active detection is close to 0 in our experiments. Those errors are caused by the missing match between the register sequence with its DFA fingerprint. The reason is that the synchrony issue among those packets, where packets are sent asynchronously without the order.

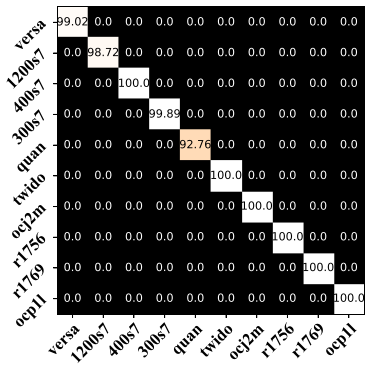


Fig. 16. F1 score of the detection via active probing.

C. Well-Designed Attack Detection

To evaluate the effectiveness of our approach in real attack detection, we simulate two attacks in experiments, including device replacement and code modification attack. Table V lists the details of those attacks and the performance of our detection approach, including precision, recall, and f1 score. The first type of attack is "device replacement attack". We simulate the replacement attack by randomly changing the device IP address among 10 devices. Specifically, we use the attack on Schneider Quantum to validate the performance of our model. The result shows that the model achieves at 95.8% F1-score.

The second type of attack is "code modification attack" which consists of "collision attack" and "mechanical offside attack". The"collision attack" is to disturb the normal execution of traffic light, which can probably result in a vehicle accident. The "Force signal on attack" modify device code and directly force the registers, which turn on "green light". Since these registers are visible, we use detection through network flows. Results show that we detect them at 95.6%. The "timing attack" and "intermediate attack" are elaborate designed based on invisible registers. They modify device code and leverage an intermediate register, which is an invisible register to conceal itself or trigger an attack at a specific time. We use the detection via active probing to identify those attacks, and results are respectively 97.3% and 99.2%. The"mechanical offside attack" is to destroy a machine by driving its mechanical device over its position threshold. The "confusion logic attack" directly changes the code to force the registers which control machine movement on when the machine gets to position threshold. The "stage attack" and "timing attack" use invisible registers. They use intermediate registers or time registers to conceal itself and will break out at a specific time or at a specific execution stage.

Furthermore, our model detects within 2s through the passive method, and 3s via the active method. In brief, our model detects them at high precision and recall. The result shows that our register-based approach is promising to detect malicious attacks to ICS manufacturing.

D. Advanced Attacks Detection

Further, we investigated the complicated ICS attacks for our *iFinger*, including "PLC blaster" [4] and "Stuxnex" [24].

TABLE V
THE RESULT OF ATTACK DETECTION

Attack type	Attack name	Attack method	Detect method	sample	precision	recall	f1 score	Device
Device replacement	Replace attack	Random replace	Passive	100	100%	92.1%	95.8%	Schneider Quantum
Code modification	Collision attack	Force signal on	Passive	396	95.6%	97.6%	95.6%	S7 1200
		Intermediate attack	Active	536	96.7%	98.1%	97.3%	S7 1200
		Timing attack	Active	1746	99.0%	99.5%	99.2%	S7 1200
	Mechanical offside attack	Confusion logic	Passive	669	99.8%	100%	99.8%	S7 300
		Stage attack	Active	1123	99.9%	99.2%	99.5%	S7 300
		Timing attack	Active	1555	99.9%	100%	99.9%	S7 300

However, the source codes of those ICS attacks are not open to the public. Hence, we only provide an initial analysis based on the iFinger.

The "PLC blaster" used a notorious ICS worm to attack PLC devices in industrial manufactures. The worm stealthily executed malicious code, which was stored at the end of the device control logic. The worm stored its variables in device registers. Once the ICS device was affected by this worm, the register state sequence of the ICS device would behave abnormally, differing from benign devices. The abnormal state sequence could not match with registers-based fingerprints, and the iFinger would detect the "PLC blaster" attack.

The "Stuxnet" tampered the control logic of the ICS device in Iran's nuclear program. It brought the significant error to outputs of the nuclear infrastructure, leading to the breakdown of physical equipment. The modification by Stuxnet caused the subtle change for the output-register state sequence. Our approach can alleviate this attack by detecting the subtle difference between the device fingerprint and its state sequences.

VI. RELATED WORK

A. ICS Attack Detection

Nowadays, industrial control systems are vulnerable to attacks, suffering the threat of invasion and damage [25]. Prior works leveraged physical information to detect malicious attacks, i.e., sensor signals [9]–[11]. Chen *et al.* [12] found that the sensor data of the cyber-physical system (CPS) keeps mutant along the time. They built a mapping between sensor signals and running programs on CPS and used it to identify the data injection by Man-in-the-middle attack. However, the physical information becomes invalid if an ICS controller is compromised. Cheung *et al.* [26] characterized the expected behavior of the SCADA system and utilized it to identify abnormal behaviors. Ali and Al-Shaer [27] detected abnormal sensors through monitoring the device meter. In contrast, we leverage the register-based approach to detect ICS attacks adapting to various industrial scenes. Zonouz *et al.* [28] analyzed the control logic in the PLC and found the malware by symbolic execution of the PLC. McLaughlin *et al.* [29] proposed a trusted safety verifier to be an anti-virus software for the PLC controller. Their works focused on offline detection for malicious attacks on ICS devices. Differing from prior works, our register-based approach would be deployed in manufacturing systems, and detect ICS attacks on the fly. Furthermore, our iFinger would subtle attacks in the ICS network.

B. Fingerprinting

The fingerprinting technique is used to identify the unique signature of a host, which has been widely studied for more than 20 years. Fingerprinting application is ranged from an operation system (OS) classification to physical device identification. OS fingerprinting is to utilize the network stack implementation difference between various OS versions and generate their signatures for classifying on the remote. Nmap [30] actively send dozens of TCP/UDP packets to remote hosts and extract header fields as fingerprints from response packets. Shamsi *et al.* [31] and Shamsi *et al.* [32] combined different time latency (TCP retransmission timeout) between online hosts and the server with header fields of the network stack for the fingerprinting operation system. In contrast, OS fingerprinting cannot be used to detect malicious attacks on ICS networks.

Like OS, device fingerprinting is to utilize invariant characteristics of devices to generate their signatures online. Formby *et al.* [33] classified the device type in the cyber-physical system based on the data process time difference between different types of devices. Li *et al.* [34] utilized the banner of packets in industrial protocols to discover online ICS devices and further provided an overview of ICS devices at large-scale. Fachkha *et al.* [35] further surveyed the cyber-physical system on the Internet. Antonakakis *et al.* [36] also utilized packet banners to discover embedded devices that were infected by the Mirai virus. Yang *et al.* [37] proposed a convolutional neural network to generate fingerprints of the embedded devices on the Internet. Furthermore, Li *et al.* [38] leveraged the difference among file systems of embedded devices to generate their fingerprints that are fine-grained product versions. The manual effort on analyzing packet banners would impede collecting numerous device fingerprints as the number of products increases. Feng *et al.* [39] proposed the commercial search engine to migrated manual efforts during device fingerprint generation. However, these device fingerprints cannot be used to detect individual devices and find malicious attacks.

Compared to coarse device information, individual device fingerprinting requires a high cost to obtain invariant characteristics. Kohno *et al.* [40] proposed monitoring to traffic

for calculating the clock skews as features for generating individual host fingerprints. Kneib and Huth [41] analyzed the controller bus protocol, and sampled the signal of the electronic control unit to extract features for fingerprinting electronic control unit. Li *et al.* [42] extracted the texture features of 3D printers to fingerprint 3D printers uniquely. However, their fingerprinting technique needs to contact with physical devices. In contrast, our register-based approach would generate individual ICS device fingerprints through remote detection on the network.

VII. Conclusion

Industrial control systems are being exposed to severe security risks, and any subtle attack will cause significant damage to the critical infrastructure. In this paper, we propose the *iFinger*, a novel approach to detect subtle ICS attacks for manufacturing systems. We utilize an observation that register status values can reflect the physical characteristics of ICS controllers. The iFinger generates device fingerprints based on register states, and use them to detect ICS attacks through either passive or active detection. We present a prototype of the iFinger and conduct real-world experiments to validate the effectiveness of our iFinger. Results show that our approach achieves 97.1% F1 score in device identification. Furthermore, we simulate two typical attacks (replacement and tempering attacks) in manufacturing systems. Our device fingerprints would detect those malicious attacks within 3s latency at 98.0% accuracy.

References

[1] H. Song, R. Srinivasan, T. Sookoor, and S. Jeschke, *Smart Cities: Foundations, Principles, and Applications*. Hoboken, NJ, USA: Wiley, 2017.

[2] I. Butun, P. Österberg, and H. Song, "Security of the Internet of Things: Vulnerabilities, attacks, and countermeasures," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 616–644, 1st Quart., 2019.

[3] A. Cardenas *et al.*, "Challenges for securing cyber physical systems," in *Proc. Workshop Future Directions Cyber-Phys. Syst. Secur.*, vol. 5, no. 1, pp. 1–7, 2009.

[4] R. Spenneberg, M. Brüggemann, and H. Schwartke, "PLC-blaster: A worm living solely in the PLC," *Black Hat Asia*, vol. 16, pp. 1–16, Jul. 2016.

[5] J. Klick, S. Lau, D. Marzin, J.-O. Malchow, and V. Roth, "Internet-facing PLCs as a network backdoor," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 524–532.

[6] S. E. McLaughlin, "On dynamic malware payloads aimed at programmable logic controllers," in *Proc. HotSec*, 2011, pp. 1–6.

[7] S. McLaughlin and P. McDaniel, "SABOT: Specification-based payload generation for programmable logic controllers," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 439–449.

[8] S. Soltan and G. Zussman, "EXPOSE the line failures following a cyber-physical attack on the power grid," *IEEE Trans. Control Netw. Syst.*, vol. 6, no. 1, pp. 451–461, Mar. 2019.

[9] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi, and M. Srivastava, "PyCRA: Physical challenge-response authentication for active sensors under spoofing attacks," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 1004–1015.

[10] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 1–33, May 2011.

[11] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: Risk assessment, detection, and response," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, New York, NY, USA, 2011, pp. 355–366, doi: 10.1145/1966913.1966959.

[12] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 648–660.

[13] *The Introduction of Siemens Product*. Accessed: 1996. [Online]. Available: https://new.siemens.com/global/en.html

[14] *Fins*. Accessed: 2019. [Online]. Available: https://automation.omron.com/en/us/

[15] *CIP*. Accessed: 2019. [Online]. Available: https://en.wikipedia.org/wiki/Common_Industrial_Protocol

[16] *Modbus*. Accessed: 2005. [Online]. Available: http://www.modbus.org/

[17] *Ge*. Accessed: 2019. [Online]. Available: https://www.ge.com/

[18] I Automation. (2018). *Market Share of Different PLCs*. [Online]. Available: https://ipcsautomation.com/blog-post/market-share-of-different-plcs/

[19] H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, *Cyber-Physical Systems: Foundations, Principles and Applications*. New York, NY, USA: Academic, 2016.

[20] International Electrotechnical Commission. (2013). *Programmable Controllers—Part 3: Programming Languages*. [Online]. Available: https://webstore.iec.ch/publication/4552

[21] C. R. Edwards, "The logic of Boolean matrices," *Comput. J.*, vol. 15, no. 3, pp. 247–253, Mar. 1972.

[22] D. B. Rawat, C. Brecher, H. Song, and S. Jeschke, *Industrial Internet of Things: Cybermanufacturing Systems*. Cham, Switzerland: Springer, 2017.

[23] *Keras. A Netual Network API*. Accessed: 2019. [Online]. Available: https://keras.io/

[24] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Secur. Privacy Mag.*, vol. 9, no. 3, pp. 49–51, May 2011.

[25] H. Song, G. A. Fink, and S. Jeschke, *Security and Privacy in Cyber-Physical Systems*. London, U.K.: Wiley, 2017.

[26] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for SCADA networks," in *Proc. SCADA Secur. Sci. Symp.*, vol. 46, 2007, pp. 1–12.

[27] M. Q. Ali and E. Al-Shaer, "Configuration-based IDS for advanced metering infrastructure," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 451–462.

[28] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated PLC code analytics," *IEEE Secur. Privacy*, vol. 12, no. 6, pp. 40–47, Nov. 2014.

[29] S. McLaughlin, S. Zonouz, D. Pohly, and P. McDaniel, "A trusted safety verifier for process controller code," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014.

[30] (2012). *Nmap, Network Secursity Scanner Tool*. [Online]. Available: https://nmap.org/

[31] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-packet OS fingerprinting," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2196–2209, Aug. 2016.

[32] Z. Shamsi, D. B. H. Cline, and D. Loguinov, "Faulds: A non-parametric iterative classifier for Internet-wide OS fingerprinting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. - CCS*, 2017, pp. 971–982.

[33] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. Beyah, "Who's in control of your control system? Device fingerprinting for cyber-physical systems," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2016.

[34] Q. Li, X. Feng, H. Wang, and L. Sun, "Understanding the usage of industrial control system devices on the Internet," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2178–2189, Jun. 2018.

[35] C. Fachkha, E. Bou-Harb, A. Keliris, N. Memon, and M. Ahamad, "Internet-scale probing of CPS: Inference, characterization and orchestration analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2017.

[36] M. Antonakakis *et al.*, "Understanding the Mirai botnet," in *Proc. 26th USENIX Secur. Symp. (USENIX Security)*, 2017, pp. 1093–1110.

[37] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Comput. Netw.*, vol. 148, pp. 318–327, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618306856

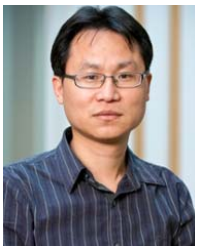
[38] Q. Li, X. Feng, R. Wang, Z. Li, and L. Sun, "Towards fine-grained fingerprinting of firmware in online embedded devices," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 2537–2545.

[39] X. Feng, Q. Li, H. Wang, and L. Sun, "Acquisitional rule-based engine for discovering Internet-of-Things devices," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, Baltimore, MD, USA, 2018, pp. 327–341.

[40] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, Feb. 2005.
[41] M. Kneib and C. Huth, "Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 787–800.
[42] Z. Li, A. S. Rathore, C. Song, S. Wei, Y. Wang, and W. Xu, "PrinTracker: Fingerprinting 3D printers using commodity scanners," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1306–1323.

Kai Yang (Student Member, IEEE) received the B.E. degree from Shandong University, China, in 2016. He is currently pursuing the Ph.D. degree with the Institute of Information Engineering, Chinese Academy of Sciences, China. His research interests include the security of the Internet of Things and Internet measurement.

Qiang Li received the Ph.D. degree in computer science from the University of Chinese Academy of Sciences in 2015. He is currently an Associate Professor at the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests revolve around mobile computing and the Internet of Things. He would like to build prototype systems based on well-founded computational models.



Xiaodong Lin (Fellow, IEEE) received the Ph.D. degree in information engineering from the Beijing University of Posts and Telecommunications, China, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Canada. He is currently an Associate Professor at the School of Computer Science, University of Guelph, Canada. His research interests include wireless network security, applied cryptography, computer forensics, and software security.

Xin Chen (Member, IEEE) received the M.S. degree from the School of Electrical Engineering, Zhengzhou University, China, in 2016. He has been an Engineer with the Institute of Information Engineering, Chinese Academy of Sciences, since July 2016. His research interest includes industrial control system intrusion detection.

Limin Sun is currently a Professor at the Institute of Information Engineering, Chinese Academy of Sciences (CAS). His research interests include the security of the Internet of Things, mobile vehicle networks, wireless sensor networks, and mobile IP. He is an Editor of the *Journal of Computer Science* and *Journal of Computer Applications*.