Report for CSAW ESC 2017

Chenglu Jin, Saeed Valizadeh, Mason Ginter and Marten van Dijk
University of Connecticut
Email: {chenglu.jin, saeed.val, mason.ginter, marten.van_dijk}@uconn.edu

***Abstract*—In this report, we briefly explain our defense strategy in order to secure Programmable Logic Controllers (PLCs).**

I. Introduction

A host-based intrusion detection (HIDS) is commonly used for monitoring specific activities and characteristics of a single device by means of software or appliance-based components known as agents [2]. A HIDS is able to provide system level checks (e.g. file integrity checking, log analysis, rootkit detection, etc.), detailed logging information, encrypted communications monitoring, and very small false positives' rates due to an ample knowledge of the configurations and characteristics of the host under its watch, in addition to having a thorough operating sytem level access to communication streams and sessions.

We have looked into different possible security proposals which could potentially be used for this project, i.e., securing the PLCs. Amongst available works, we found [3] and [4] more relevant and applicable for this project. [3] proposes a host-based defense mechanism called Symbiotic Embedded Machines or "SEM" which adds an intrusion detection functionality to the firmware layer of an embedded system. As the SEM structure operates alongside the native OS of the embedded device and not within it, it can inject generic defensive payloads into the target device regardless of it's original hardware or software. On the other hand, [4] introduces an interesting tool called PillarBox for the purpose of securely relaying Security Analytics Sources (SASs) data to combat an adversary taking advantage of an advanced malware who can undetectably suppress or tamper with SAS messages in order to hide attack evidence and disrupt intrusion detection. Pillarbox provides two unique features:

- Integrity: By securing SAS data against tampering, even when such data is buffered on a compromised host within an adversarially controlled network.
- Stealthiness: By concealing SAS data, alert-generation activity and potentially even alerting rules on a compromised host, thus hiding select SAS alerting actions from an adversary.

In this regard, in this project, by using the idea introduced in [4], we implemented a very lightweight HIDS module on the OpenPLC framework in order to keep track of input/output operations on the device. To locate signs of likely security related incidents, the HIDS's agent installed on each PLC, which we call it the *Snapshotter*, logs all the events on the controller (more specifically, input/output operations). Then the Snapshotter sends the snapshot, which is indeed the log of operations happening on the PLC at a specific time, in a *stealthy and secure way* to a server periodically for the purpose of anaylsis and intrusion detection.

On the other hand, the implemented server first checks the integrity of the log itself (in case, the attacker has already compromised the device). Moreover, it has a predefined set of acceptable input/output pairs for each device and checks and the validity of the operations happening on the controller. If any of the previous incidents happens, i.e., whether the log's integrity check fails, or an operation is detected as invalid, a flag will be raised and an intrusion is indeed captured[1]. The server takes proper actions consequently which could be terminating the controller, revoking it from the network, or even recovering it to a clean/safe state.

In order to have tamper-resistant logs, we use a forward-secure integrity protection mechanism in which the controller generate new keys after every alert generation and delete keys immediately after use [4]. For the operation validation purposes, the server actually traces deviations from predefined acceptable normal profile activities of the input/output operations. Such profile activities could be generated based on the specific applications that the PLC is being used for. As an example, lets say the adversary somehow has root access to the system, and can generate some sort of malicious input/output pairs, such activity can be captured by our proposed method, since the adversary can not tamper the logs and the server will restore the PLC into its normal state by uploading a known valid logic into the PLC.

Our defense mechanism can be summarized in security related information gathering and secure logging, sending the logs to a server for the purpose of analysis, incident identification and taking effective actions by the server to foil such incidents. We explain each part in more detail in the following sections.

II. Proposed Defense Scheme

A host-based intrusion detection (HIDS) is commonly used for monitoring specific activities and characteristics of a single device by means of software or appliance-based components known as agents. In this regard, in this project, we propose a very lightweight HIDS module on the OpenPLC framework in order to keep track of input/output operations on the device.

[1]Assuming no errors in the controller functionality itself. Note that, even if there is an error in the PLC functionality, our proposed method can capture it as well with the same procedure

To locate signs of likely security related incidents, the HIDS's agent, which we call it the *snapshotter* agent, essentially, takes a snapshot of the input/output values together with the current time stamps on each PLC, encrypts the log, and sends them *in a stealthy secure way* to a server periodically. Forward security is achieved in the key management of the encryption key. Next key is updated to the hash of the current key after each encryption. Therefore, even the adversary compromised the device completely, meaning that the encryption key is also exposed to the adversary, he is still not able to tell from the previous encrypted logs that whether he is caught by the intrusion detection systems or not.

Upon receiving the reports from PLCs, the server can use the received input and known PLC logic to simulate the expected output of the PLC. If the received PLC output does not match with the expected (simulated) output, then the server will conclude that this PLC has been compromised, so it should be restarted and reprogrammed by a correct logic program. Of course, for a specific application, the server can have a predefined set of acceptable input/output pairs for each device. This will reduce the computation of the server side. Besides the input/output behaviors, the integrity of the log can be checked at the server, in case that the attacker has already compromised the device.

Our defense mechanism can be summarized in security related information gathering and logging, incident identification, and taking effective actions to foil such incidents.

This solution has five features that make our proposed scheme very suitable to the intrusion detection for PLCs:

1) We can guarantee the integrity of the log generated by each PLCs, because we are able to verify its integrity at the server side, and we can detect any dropped package in the communication.
2) The log is sent in a stealthy way, such that the attacker is not able to tell whether he gets detected or not. This gives more advantages to the defenders to record more behaviors of the attackers, which can be used for further investigations.
3) This approach requires no redundant PLCs for detecting an intrusion, so it saves the cost of extra PLCs for implementing any intrusion detection schemes, e.g. comparing the outputs of two redundant PLCs.
4) Our approach can be applied on any legacy PLCs, because we only record the inputs and outputs of each PLCs in the log. No internal states of the PLCs are required.
5) Since the logic running on the PLCs is usually very simple, a powerful server can easily simulate a large number of PLCs in parallel. This makes our solution easily scalable to monitor a large number of PLCs.
6) In the scenario of an industrial control system, all the devices are connected using cables, so the connectivity of each PLC device is very well established. This will reduce the false positive of our system caused by the network delay.

A. Information Gathering

In order to achieve the stealth logging, we suggest to use the logging mechanism in [1]. The log has two requirements:

1) All the logs are encrypted, and the encryption key is updated in a forward secure key.
2) The logs are buffered in the device itself, and is sent periodically to the server.

The first requirement prevents the adversary, who eavesdrops the communication between the PLCs and the server, from figuring out whether the current attack is being detected or not. A forward secure key management scheme guarantees that even when the current key is compromised by the adversary, the adversary is still not able to decrypt all the encrypted logs sent before.

The second requirement prevents the attackers from dropping all the encrypted logs that possibly record some traces of the adversary, because if the report of one device is not sent to the server on time, the server will conclude that this device has compromised by attackers.

B. Incident Identification

If any of the incidents happen, i.e., whether the log's integrity check fails, or an operation is detected as incorrect, a flag will be raised and an intrusion is indeed captured[2]. The server takes proper actions consequently which could be terminating the controller, revoking it from the network, or even recovering it to a previous clean/safe state.

C. Mitigation

Once an incident has been identified, one can restart the compromised PLCs, and reprogram it to a safe/correct state/logic. If the monitoring server is not able to reset the PLC to a clean state, then a technician must be sent to physically approach the PLC and fix it. In the meanwhile, another uncompromised PLC can be deployed to replace the compromised PLC to continue the industrial process.

III. Implementation

A. Algorithms

We implemented the stealthy logging mechanism in OpenPLC framework on a Raspberry Pi. The operations in one scan cycle can be described in Algorithm 1. In OpenPLC framework, **Read_Input** function and **Write_Output** function has been modified to return 1 if any value changes in the current scan cycle. Therefore, only when the input or output values change, this event will be recorded in the log. A single event is encrypted and its key is updated to its hash value, such that the old key for encrypting this event will be overwritten right after each encryption.

Since the input or output of PLC does not change very often, only reporting changes of input/output values in the log, we can significantly reduce the size of the log. For an application

[2]Assuming no errors in the controller functionality itself. Note that, even if there is error in the PLC functionality, our proposed method can capture it as well with same procedure

Algorithm 1 PLC Logging

```
procedure LOGGING(Period)
    EL = {}
    T = 0
    Event_ID = 0
    Update = 0
    while True do                                   ▷ Scan Cycle
        Update = 0
        T = T + 1
        Update = Read_Input()
        PLC_Logic()
        Update = Write_Ouput()
        if Update == 1||T mod Period == 1 then
            TempLog = {T, Input, Output, Event_ID}
            EL ←(ENCRYPT(Temp_Log, Key))
            TempLog = ∅                             ▷ Delete the plaintext
            Key =HASH(Key)
            Event_ID = Event_ID + 1
        end if
        if T mod Period == 0 then
            SEND(EL)
            EL = ∅                                  ▷ Empty the buffer
            Event_ID = 0
        end if
    end while
end procedure
```

Algorithm 2 Server Monitoring

```
procedure MONITORING(Period)
    Alarm = False
    Last_Rec_T = Current_T
    while True do
        while True do
            if Current_T − Last_Rec_T > Period then
                Alarm = True
                Break                               ▷ Missing one Packet
            end if
            if EL ← REC_PACKET( )then
                Last_Rec_T = Current_T
                Break                               ▷ Receive one Packet
            end if
        end while
        (L, Key) ← DECRYPT_PACKET(EL, Key)
        Alarm ← VERIFY_FORMAT(L)
        (T′, Input′, Output′, Event_ID′) ← L
        (T″, Output″) ← PLC_SIMULATE(T′, Input′)
        Alarm ← COMPARE(T′, Ouput′, T″, Output″)
        if Alarm then
            Restart PLC
        end if
    end while
end procedure
```

that requires the inputs/outputs of PLCs to be updated very frequently, we suggest to reduce reporting period or ignore some small fluctuations in the analog output.

After a predefined reporting period is elapsed, the PLC needs to send all the buffered encrypted log to the server and empty the buffer for the next period.

The algorithm on the server side works as Algorithm 2. The server waits for the incoming packet from the PLC. If the packet is not received on time, the server concludes that the PLC is compromised, we need to restart the PLC. If the server gets the packet on time, then we need to use the associated key to decode this packet and update the key stored at the server. After that, we need to check whether the decrypted data has the correct data format or not. If not, then it means the integrity of the packet has been compromised, we need to restart the PLC. If the integrity is also valid, then we need to extract the input change in this period with its time stamp. This input change and time information is applied to the PLC logic simulator, which will generate the expect output change with its time stamp. Then this expected output is compared with the output received from the PLCs. If they do not match, then the server knows that the logic running on the PLC has been maliciously modified. Therefore, it is also required to restart the PLC in this case.

B. Implementation Details

As cryptographic primitives, we use AES-128 and SHA-256 as the encryption algorithm and hash function respectively. The updated key is the first 16 bytes of the hash value of the previous key. The reporting period is set to 10 seconds in the prototype implementation, but it can be easily adapted to a different value if it is required by the application.

To minimize the size of the log we sent, we design one data format to record one event (input or output change) in less than 128 bits, which can fit in one encryption block of AES. The data format is depicted in Figure 1. The first byte is used as an indicator of the start of one event; we set it as 0xFF. The second and third bytes are used to store the event ID in the current time period. Since the PLC is required to run on 100Hz, and we set the reporting period to be 10 seconds, the maximum of events can happen in one period is 1000. Therefore, two bytes are more than enough to represent the maximum number. The forth and fifth bytes are reserved for device ID. In total, 65536 devices are allowed to be managed by one server. The following 4 bytes are used to store the time stamp, because in OpenPLC framework, it is a 4 byte variable. The next six bytes are divided for storing digital inputs, digital outputs and analog output value respectively. Each of them takes two bytes. Since the number of digital inputs and digital outputs on a Raspberry Pi is 14 and 11. Two bytes are enough to store all the input/output pins. Also, only one analog pin can be used on a Raspberry Pi, and this value can be represented by a 16-bit value stored in 2 bytes. In the end, another byte of 0xFF is appended to indicate the end of one event log.

#Byte	1 Byte	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
	Start	Event ID	Device ID	Time	Digital Inputs	Digital Outputs	Analog Outputs	End
Example	0xFF	0x0002	0x1234	0x00000010	0xC000	0x8000	0x7832	0xFF

16 Bytes in total

Fig. 1. Data format of one event in the log.

IV. Evaluation

In this section, we will evaluate our solution and implementation in terms of security and performance.

A. Security Analysis

Our adversarial model is that we assume the adversary has the capability to upload malicious logic to the PLC that will generate erroneous outputs or further compromise the entire PLC.

In addition, we assume that the logging mechanism is working properly until a certain point after the compromise initiates. In other words, we assume there exist a time window that the attacker has initiated the intrusion, but the logging is still working properly. The overall timeline can be illustrated as Figure. 2.

B. Performance Overhead

V. Future Works

In the future, we can implement this scheme to a complete industrial control network, which has a real powerful server connected to. This can help us study how much this scheme can be scale in practice, due to the constraints on the network bandwidth or a certain application.

Also, we need to try to find a more efficient mitigation strategy, that affects the real industrial process as little as possible.

VI. Conclusion

In this project, we implemented stealthy logging mechanism on the PLCs, and the server also runs the monitoring program.

Acknowledgment

The authors would like to thank Thiago Alves for the helpful discussion.

References

[1] Bowers, Kevin D., et al. *Pillarbox: Combating next-generation malware with fast forward-secure logging.*, 3rd ed. International Workshop on Recent Advances in Intrusion Detection. Springer, Cham, 2014.

[2] Scarfone, Karen, and Peter Mell. *Guide to intrusion detection and prevention systems (idps).*, NIST special publication 800.2007 (2007): 94.

[3] Cui, Ang, and Salvatore Stolfo. *Defending embedded systems with software symbiotes.* Recent Advances in Intrusion Detection. Springer Berlin/Heidelberg, 2011.

[4] Bowers, Kevin D., et al. *Pillarbox: Combating next-generation malware with fast forward-secure logging.* International Workshop on Recent Advances in Intrusion Detection. Springer, Cham, 2014.

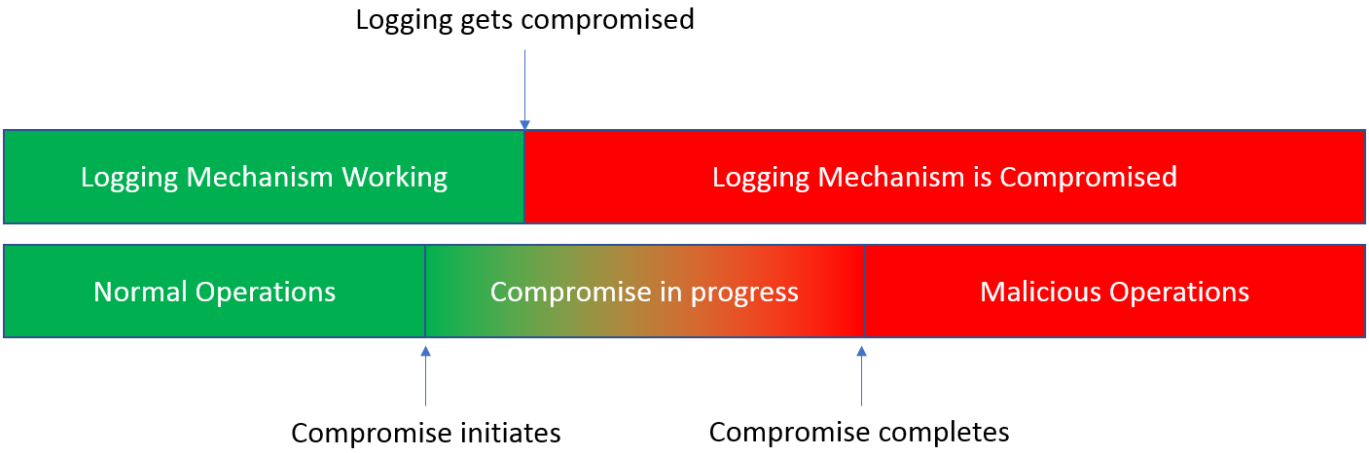

Fig. 2. Time line of one compromise.