Security of PLCs with Raspberry Pi

1st Vignau Benjamin
Student in Sécurité et Technologies Informatique
Institut National des Sciences Appliquées Centre Val de Loire
Bourges, France
benjamin.vignau@insa-cvl.fr

2nd Kawalec Joseph
Student in Sécurité et Technologies Informatique
Institut National des Sciences Appliquées Centre Val de Loire
Bourges, France
jospeh.kawalec@insa-cvl.fr

3rd Ronteix–Jacquet Flavien
Student in Sécurité et Technologies Informatique
Institut National des Sciences Appliquées Centre Val de Loire
Bourges, France
flavien.ronteix-jacquet@insa-cvl.fr

4th Mace Lorin
Student in Security Technologies Informatique
Institut National des Sciences Appliquées Centre Val de Loire
Bourges, France
lorin.mace@insa-cvl.fr

5th Briffaut Jérémy
Assistant Professor at Laboratoire d'Informatique Fondamentale d'Orléans
Institut National des Sciences Appliquées Centre Val de Loire
Bourges, France
jeremy.briffaut@insa-cvl.fr

***Résumé*—As the new CSO team of CannotPwn Factory, we have to build a new system to control the factory. The last team failed to handle a cyber attack. In result our company lost a lot of money, and we have to built a new system more resilient.**
Today, the PLCs are more present in factory, and control a lot of criticals infrastructures. But, this systems are really weak, so they are a great target to reduce productivity of industries or plant.
The biggest example of cyber attack against PLC's is stuxnet. This program set up a huge "man in the middle" attack and corrupt data received and send to the PLCs. In result, stuxnet was able to distort the comportment of PLCs and deteriorate the centrifuge of Iranian power plants.

I. Introduction

In this challenge, we will try to develop our solution to prevent, detect, and mitigate attacks such as stuxnet, against PLCs. In a first part, we will describe our solution to protect the system. After that, we will present our methods to test our solution.
Our solution will be based on several ways to detect intrusion, such as SIEM, powerful logs systems and correlators, integrity check, real time comportment analysis. Of course all actions will be logged, and if an error is considered too relevant, a warning or a report will be send to the administrator.
To mitigate the attacks, we will correct minors and medium errors thanks to a determinist behavior of PLCs. We will also program a hard reset, that could allow us to wipe a PLCs program and completely set up a new one, fully functional. This hard reset will be used only if we detect a highly corrupted systems.
We will also set up some honey pots to handle and analyse some attacks. This will be useful to adapt our systems against the new types of cyber attacks.
To test our solution, we will set up somes attacks, like leaking critical informations (for fingerprinting), authentication method bypass, replay attack, fuzzing, and alter the behaviour of the PLCs withs some calculated errors.

II. Cyber attack detection

A. Logs systems

The first part of our solution is based on a huge log system. This will log all the input and output of each GPIO of each PLC. This system will be really useful with a behavior model of the PLC, so it could compare reality and models, and detect changes in the PLC behavior. Moreover, the log system could also help to create betters models. We could use the logs to detect some performance issues, and also, in case of malicious inputs, we will be able to understand the behavior that the attackers wanted to induce in the PLC. In the end, if an attack succeed, an huge examination of the logs could permit a better understood of the attack and will help to detect others. For the log system we will use a SCADA system. We are still hesitating for which software we will use. However, because of it's documentation (and because it is open-source) OpenSCADA seems to be, for us, the best choice. This system will allow us to make our own rules to prevent and detect cyber attacks.

B. SIEM, correlators, IA and kernel built

Another way to detect intrusions in our system will be to set up watchdogs, SIEM, and to create our own Artificial Intelligence (AI) to warn the CSO. For that we will re-compile our kernel with some flags to increase his security. Thanks to

whitelist of users and actions, our AI will define a probability of corrupted system. In case of a too big probability, the CSO will be warned and the PLC will be flushed and re-setuped. But before flushing the PLC, we will save it to analyse the attack and still upgrade the AI. Moreover, we will develop a honeypot for protect our system and always learn new ways of attacks to protect the systems against them.
Of course, our AI will not control the PLCs. Our AI is set up to detect systems intrusions such as rootkit, trojans, or corrupted programs.
We will compile our kernel to include grsecurity components and impact as few as possible the performances of the system. Thanks to it, we will be able to create our own roles and access to criticals files such as /etc/passwd or the PLCs programs.
Moreover, we will develop our own probe and correlators thanks to the framework prelude SIEM.

C. Behavior changes

The major section of our solution is based on behavior analysis. For that, we will define a deterministic behavior graph. In this, we will define all the possible states of a PLC, and all the transitions. Each state will be a function that take the inputs in arguments and could only send output to control equipments or alert as result. Moreover, we could define somes messages with the inputs tension of each GPIO and define specifiques intervals for each transition. For example, we could define a state A, and the transition to the state B could be 1 Volt, more or less 0.1V, so [0.9V,1.1V]. That will increase resilience to physical issues.
Thanks to it, we will be able to control all the inputs and outputs (I/O) of a PLC. All the I/O will be duplicate, and a PLC, with only a connection to a private network (dedicated to alert CSO) will have to control the correct behavior of the other PLC. Thanks to it, if the PLC who is controlling the equipment is corrupted by a program such as stuxnet, the second PLC will detect a bad behavior and alert the CSO.

III. Attacks isolations

A. Sandboxing

To increase the resistance of our solution, we have decided to isolate all the PLCs and their components. Each of them will have his own running environment. Again, we will do that with Grsecurity and the PaX component. They will allow us to define roles and authorize the programs to run only what they have to do.
Moreover, thanks to the file system hardening (especially the chroot hardening system and the trusted path execution) the PLCs will be emulated run in their own environment. So, if one is corrupted, he will not corrupt the others. Moreover, Grsecurity permit to prevent direct userland access by kernel, and hide other process to unprivileged users. This will also be useful to isolate a detected attack on the system.

IV. Mitigation

A. Errors correction with beahavior prediction

With a determinist behavior, we can wait for only few values. We can determine all the authorized inputs and outputs for each state. We can make a dynamic whitelist of the inputs, by finding the right next state and we can correct input values if it's needed.
For example (see Behaviour graph), your PLC is in state E, so you know that the only possibilities are to go to state B or C. admit you need a 4V input on GPIO1 to go to B and 2V to go to C. If the PLC receive an 1.5V in GPIO1, he will automatically act as if he had received a 2V input. Moreover, each state must be programed to have a default behavior.
This will be used when the input values is too far away of authorized inputs and the error could not be corrected. Back to our example, if the input values is 0.5V, the PLC need a default behavior such as ignore this value or go into an "error state" that will be defined in our error management.

B. Hard reset

If one of PLCs is detected as corrupted, we will make a hard reset. That is to say, that we will flush all his programs and re setup totally all the PLC. For that we will use the Union File System that allow us to mount two partitions. One of them is in read/write (rw) mode and the other is in the read only (ro) mode. If the PLC is detected as corrupted, all the data in the ro partition will replace the rw partition that is corrupted.

V. Test of our solution

We will test our solution with automated script to avoid regression. We will make three kinds of tests.
First of all, for the attacks which are coming from GPIO pins, we will try to fuzze the passed values at a defined ratio, to improve our error detection and correction system. We will also try to isolate request and try to replay them. It is in our objective to test our raspberry in the worst case scenarios in order to handle a maximum of perturbations.
In a second time, we will search for numerous vulnerabilities on the other side of the infrastructure, like leaking critical informations, bypassing authentication mechanisms, replaying some requests, exploiting security misconfigurations, code injections, race conditions and many others...
Finally, it is really important in this context to test the defence in depth of our solution. We will test the isolation of our system, with the privileges of critical parts and process of our architecture, the reactivity and accuracy of our system for detecting a corrupted PLC and to reset it. The accuracy is really important here because if we can trigger hard resets a lot, this can be viewed as a DDOS attack

VI. Conclusion

To conclude, our solution will be based on a behavior analysis of the PLCs and the systems. We will develop our own PLC behavior and look up for intrusions. Each issues will be logged and if too importants errors are detected, the CSO will be warned.

Our solution will impact as few as possible the system, because the majority of the systems rules will be implemented in the kernel. Moreover we will develop our own AI to detect behavior changes, and we will try to optimize it as much as possible. Our solution used a lot of framework, so we will not construct all from scratch, but we will made a unique system, dedicated to this problem.
Of course, we will test our systems by a lot of attacks, some of them will be known, but we will also try to develop new way to attack our solution.

VII. And after ?

In our first try to conceive a system, we were somewhat ambitious with a solution that had 3 raspberry pi. The heart of the solution was to separate physically 3 raspberry with 3 different function.
The first one, the system core, had 2 interfaces, one to the PLC and one with the outside in order to control the whole system. The first detection of attack attempt is with an AI that will detect a suspicious behavior from the outside commands like the manipulation of some files,... In this case, like in our proposition of solution, the raspberry had a deterministic behavior with the PLC if there has been no corruption.
To detect a bad behavior, we have an other raspberry with the same initial configuration but without any connection to a network, so we can be sure that this one can't be corrupted by a malware. The main idea is that the PLC will send to the 2 raspberry, the main reply with an instruction and the second will compare the main's answer with its own answer. if the 2 are too different, the second is going to hard reset the main. At the hard reset of main, a backup raspberry (the third) will completely reinstall the system.
This is a vision to the future because in addition to protect the system with the protections mentioned above, we separate physically (and from the network, source of attack) the protections, with backup and detection system on other raspberry. Moreover we could made a huge server that will analyse all the data from all the PLCs and always improve our solution.

VIII. Figures and Tables

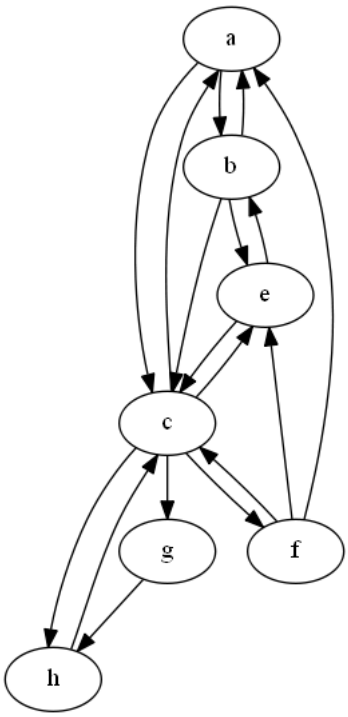

Figure 1. Behavior state.

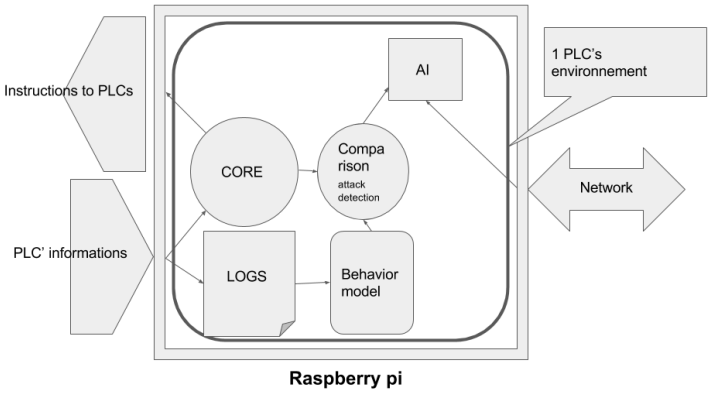

Figure 2. Logical components of each PLC.