On Dynamic Malware Payloads Aimed at Programmable Logic Controllers

Stephen McLaughlin
Penn State
smclaugh@cse.psu.edu
SCADA and PLCs

- PLCs are the lowest level of computation in the SCADA system
Stuxnet’s PLC payload

Stuxnet delivered a *precompiled payload*. The specifics details of the target had to be known ahead of time.

Against any other target, the payload would have random or no affect.

Are dynamic payloads for unknown or partially known targets possible?
An engineering problem

- Writing malware to overcome the obscurity of process control systems is an engineering problem.

- This problem can be solved, as can all software engineering problems, through a breaking down into modular steps.
Code reuse

Ideally, the adversary need only specify the payload goal.

3.1 Payload Goals
A payload goal specifies the behavior that the adversary wishes to cause in the plant: It may be as simple as "Open all breakers in the electrical substation," or as complex as "Identify all incompatible regions of track and signal two trains to enter a conflicting route:" It may also be very broad in scope, e.g., "Identify and violate all safety checks maintained by the PLC:" Regardless of the exact goal, the dynamic payload will ultimately be a sequence of one or more assignments to output variables that achieves the goal in the plant: Thus, the payload goal can be thought of as a template for the dynamic payload, with the specifics being filled in by the steps of process analysis and payload generation.

3.2 Process Analysis
The job of process analysis is to convert the logic and data read from a PLC into a canonical process representation: A complete process representation should contain both a canonicalization of the PLC code, and the mapping from input and output variables to their corresponding sensors and devices in the plant: While the code can always be obtained by reading the PLC's function blocks, it may not be possible to obtain or infer the device mapping: The challenges associated with each task are described as follows:

Recovering the Boolean equations.

The first step towards obtaining the process representation is to recover the set of Boolean equations $\Phi$ that represents the logic: Similar to the procedure for reverse engineering a typical computer program, the native code will have to be disassembled into mnemonics and then transformed to...
The format library contains platform-dependent disassemblers and device IDs.

Because the payload was precompiled, it is believed that Stuxnet's authors had previous knowledge of the exact layout of the target process and plant. Thus, it is unlikely that its attack would succeed against any other facility besides the intended one. This need for a priori knowledge of the target is assumed to be the main mitigating factor against the more common occurrence of PLC malware. It is this belief that motivates our inspection of dynamic payload generation for PLC malware.

Dynamic Payloads

Figure shows the basic steps PLC malware take to dynamically construct a payload against an unknown process. As with any malware, it must first infect one or more hosts before executing its payload. Infection may occur via viral propagation, Trojan horse, insiders, or any other attack vector. PLC malware ultimately tries to infect a host that can reach a PLC. Because the details of the process are unknown at this point, a payload cannot yet be directly uploaded. Instead, the PLC's memory contents are read for a step called process analysis, which produces a canonical process representation. This may require the use of a format library to decode proprietary binary formats. The process representation is then used by the subsequent payload generation step to create a payload that will achieve the payload goal.

For the remainder of this section, we describe techniques by which each of the above steps may be achieved:

3.1 Payload Goals

A payload goal specifies the behavior that the adversary wishes to cause in the plant. It may be as simple as "Open all breakers in the electrical substation," or as complex as "Identify all incompatible regions of track and signal two trains to enter a conflicting route:" It may also be very broad in scope, e.g., "Identify and violate all safety checks maintained by the PLC:" Regardless of the exact goal, the dynamic payload will ultimately be a sequence of one or more assignments to output variables that achieves the goal in the plant. Thus, the payload goal can be thought of as a template for the dynamic payload, with the specifics being filled in by the steps of process analysis and payload generation.

3.2 Process Analysis

The job of process analysis is to convert the logic and data read from a PLC into a canonical process representation. A complete process representation should contain both a canonicalization of the PLC code, and the mapping from input and output variables to their corresponding sensors and devices in the plant. While the code can always be obtained by reading the PLC's function blocks, it may not be possible to obtain or infer the device mapping. The challenges associated with each task are described as follows:

Recovering the Boolean equations.

The first step towards obtaining the process representation is to recover the set of Boolean equations \( \Phi \) that represents the logic. Similar to the procedure for reverse engineering a typical computer program, the native code will have to be disassembled into mnemonics and then transformed to

[Diagram: Flowchart showing the process of constructing a dynamic payload, including steps for Payload Goal, Process Representation, Payload Generation, and Delivery Mechanism.]
Logic programming

• Logic programs simulate Boolean circuits.

• A PLC program maps a set of input variables \( \mathcal{I} \) to a set of output variables \( \mathcal{O} \).

• Values for \( \mathcal{I} \) are received from the sensors in the plant, and values in \( \mathcal{O} \) are sent to the plant to manipulate devices.

• A set of internal state variables \( \mathcal{C} \) and timer variables \( \mathcal{T} \) are also available.

• A logic program is a set of expressions \( \Phi \) s.t.

\[
\forall (y \leftarrow \phi) \in \Phi, \text{Var}(\phi) \subseteq \mathcal{I} \cup \mathcal{O} \cup \mathcal{C} \cup \mathcal{T} \quad \text{and} \quad y \in \mathcal{O} \cup \mathcal{C} \cup \mathcal{T}
\]

• Note that in practice, we can often differentiate the four types of variables.
The problem of autonomously generating a PLC payload is broken down into several steps. First, the malware must break the network security mechanisms of the MTU to allow for uploading new code. Once executing within the MTU, the payload must have a mechanism to spread to other PLCs in the SCADA system that are directly connected to the MTU. This can happen using code signed by a legitimate vendor as detailed in [8].

Stuxnet had a sophisticated infection process which used code signed by two valid certificates, and Microsoft Windows zero-day exploits. The goal was to spread from an infected PLC to other vulnerable PLCs in the network.

Recent concerns surrounding PLC malware were addressed with a summary of PLC functionality. PLCs are computers used to control plant devices to keep the process functioning correctly. The logic may be detailed after the summary. The logic to produce a set of output variables is often referred to as the control program. The control program is usually directly written in the language that the PLC understands, but it can also be written in a high level programming language and executed on the PLC.

Program execution on PLCs differs substantially from on general-purpose computers. A PLC program, once in the logic, is read from the sensors in the plant and processed by the logic to produce a set of output variables that control the physical devices in the plant. Separate memory areas are used for the logic and the state of the physical devices.

The logic may also maintain a set of internal state variables to represent the current state of the physical devices and the process. It is the state of these internal state variables that is checked against clues to ensure the process is proceeding in a safe state.

Surveying examples of malware affecting SCADA systems, there are several previous examples of malware having affected industrial control plants. The problem of autonomously generating a PLC payload is broken down into several steps. First, the malware must break the network security mechanisms of the MTU to allow for uploading new code. Once executing within the MTU, the payload must have a mechanism to spread to other PLCs in the SCADA system that are directly connected to the MTU. This can happen using code signed by a legitimate vendor as detailed in [8].

Stuxnet had a sophisticated infection process which used code signed by two valid certificates, and Microsoft Windows zero-day exploits. The goal was to spread from an infected PLC to other vulnerable PLCs in the network.

Recent concerns surrounding PLC malware were addressed with a summary of PLC functionality. PLCs are computers used to control plant devices to keep the process functioning correctly. The logic may be detailed after the summary. The logic to produce a set of output variables is often referred to as the control program. The control program is usually directly written in the language that the PLC understands, but it can also be written in a high level programming language and executed on the PLC.
A pedestrian crossing

Pedestrian green and red (output)

Pedestrian green and red (output)
A pedestrian crossing

Request to cross button (input)

A pedestrian crossing

Traffic

Pedestrian

Signal Box

Output

Input

pg, pr

tg, ty, tr

pressed
A pedestrian crossing

\[(y \leftarrow \phi) = (p_g \leftarrow \text{pressed} \land t_r), \quad \text{Var}(\phi) = \{\text{pressed}, tr\}\]
Process analysis

Because the payload was precompiled it is believed that Stuxnet’s authors had previous knowledge of the exact layout of the target process and plant. Thus, it is unlikely that its attack would succeed against any other facility besides the intended one. This need for a priori knowledge of the target is assumed to be the main mitigating factor against the more common occurrence of PLC malware. It is this belief that motivates our inspection of dynamic payload generation for PLC malware:

3 Dynamic Payloads

Figure - shows the basic steps PLC malware take to dynamically construct a payload against an unknown process: As with any malware, it must first infect one or more hosts before executing its payload. Infection may occur via viral propagation, Trojan horse, insiders, or any other attack vector. PLC malware ultimately tries to infect a host that can reach a PLC. Because the details of the process are unknown at this point, a payload cannot yet be directly uploaded. Instead, the PLC’s memory contents are read for a step called process analysis, which produces a canonical process representation. This may require the use of a format library to decode proprietary binary formats. The process representation is then used by the subsequent payload generation step to create a payload that will achieve the payload goal in the plant. If payload generation is successful, then a payload tailored to the specific process may be uploaded to the PLC and executed.

For the remainder of this section, we describe techniques by which each of the above steps may be achieved:

3.1 Payload Goals

A payload goal specifies the behavior that the adversary wishes to cause in the plant. It may be as simple as “Open all breakers in the electrical substation” or as complex as “Identify all incompatible regions of track and signal two trains to enter a conflicting route.” It may also be very broad in scope, e.g., “Identify and violate all safety checks maintained by the PLC.” Regardless of the exact goal, the dynamic payload will ultimately be a sequence of one or more assignments to output variables that achieves the goal in the plant. Thus, the payload goal can be thought of as a template for the dynamic payload, with the specifics being filled in by the steps of process analysis and payload generation.

3.2 Process Analysis

The job of process analysis is to convert the logic and data read from a PLC into a canonical process representation. A complete process representation should contain both a canonicalization of the PLC code, and the mapping from input and output variables to their corresponding sensors and devices in the plant. While the code can always be obtained by reading the PLC’s function blocks, it may not be possible to obtain or infer the device mapping. The challenges associated with each task are described as follows:

Recovering the Boolean equations. The first step towards obtaining the process representation is to recover the set of Boolean equations \( \Phi \) that represents the logic. Similar to the procedure for reverse engineering a typical computer program, the native code will have to be disassembled into mnemonics and then transformed to...
Process analysis: PLC code

• How do we obtain the canonical process representation from the binary logic?

• While PLC ISAs differ between vendors, many implement the accumulator based architecture specified by the IEC 61131-3 *Instruction List* (IL) language.

• Thus, converting code to a canonical format of Boolean expressions requires two steps:

```
Binary code → IL code → Boolean expr’s
```

  Disassembly: Using a mapping from the format library.

  Logic recovery: We have implemented in < 200 lines of Standard ML.
Process analysis: configuration data

- PROFINET and PROFIBUS both network PLCs to devices with some onboard intelligence.
- Each PROFI* compliant product has a unique ID that can be queried.
- Stuxnet looked for centrifuge IDs.
- A database of such IDs can be used to map logic variables to physical devices.
- PROFI* device IDs can be scraped from reseller product list, .GSD files, and profibus.com.
- Collect them all!
Payload generation

Because the payload was precompiled, it is believed that Stuxnet's authors had previous knowledge of the exact layout of the target process and plant. Thus, it is unlikely that its attack would succeed against any other facility besides the intended one. This need for a priori knowledge of the target is assumed to be the main mitigating factor against the more common occurrence of PLC malware. It is this belief that motivates our inspection of dynamic payload generation for PLC malware.

Figure 3.1 shows the basic steps PLC malware take to dynamically construct a payload against an unknown process. As with any malware, it must first infect one or more hosts before executing its payload. Infection may occur via viral propagation, Trojan horse, insiders, or any other attack vector. PLC malware ultimately tries to infect a host that can reach a PLC. Because the details of the process are unknown at this point, a payload cannot yet be directly uploaded. Instead, the PLC's memory contents are read for a step called process analysis, which produces a canonical process representation. This may require the use of a format library to decode proprietary binary formats. The process representation is then used by the subsequent payload generation step to create a payload that will achieve the payload goal in the plant. If payload generation is successful, then a payload tailored to the specific process may be uploaded to the PLC and executed.

For the remainder of this section, we describe techniques by which each of the above steps may be achieved:

3.1 Payload Goals
A payload goal specifies the behavior that the adversary wishes to cause in the plant. It may be as simple as "Open all breakers in the electrical substation," or as complex as "Identify all incompatible regions of track and signal two trains to enter a conflicting route." It may also be very broad in scope, e.g., "Identify and violate all safety checks maintained by the PLC." Regardless of the exact goal, the dynamic payload will ultimately be a sequence of one or more assignments to output variables that achieves the goal in the plant. Thus, the payload goal can be thought of as a template for the dynamic payload, with the specifics being filled in by the steps of process analysis and payload generation.

3.2 Process Analysis
The job of process analysis is to convert the logic and data read from a PLC into a canonical process representation. A complete process representation should contain both a canonicalization of the PLC code and the mapping from input and output variables to their corresponding sensors and devices in the plant. While the code can always be obtained by reading the PLC's function blocks, it may not be possible to obtain or infer the device mapping. The challenges associated with each task are described as follows:

Recovering the Boolean equations. The first step towards obtaining the process representation is to recover the set of Boolean equations $\Phi$ that represents the logic. Similar to the procedure for reverse engineering a typical computer program, the native code will have to be disassembled into mnemonics and then transformed to...
Inferring device types

• Will not always be possible to learn devices from PROFIBUS, PROFINET, etc.

• However, if the class of plant under attack is known, certain domain specific invariants will link variables of interest.

• For example:
  ‣ A time delay of a few seconds is enforced before a motor can reverse directions.
  ‣ Electrical substation switchgear state changes must be executed in specific orders.

• Of course, this requires that the adversary have some domain specific knowledge of the target, but no target specific knowledge is needed.
Safety interlocks

- Safety interlocks are invariants over the outputs of a control system that must never be violated

- Pedestrian crossing interlock:
  - Let $p_g$ and $t_g$ be the Boolean variables for the pedestrian green light and the traffic green light respectively
  - Regardless of the particulars of the light scheme, the following must hold:
    \[-(p_g \land t_g)\]
  - May be explicit: The property is stated as a check in the logic
  - Or implicit: The property is never violated by the logic
Exploiting safety interlocks

- To exploit a safety interlock, the malware must
  1. Extract the interlock
  2. Find an assignment to some subset of $\mathcal{O}$ that violates the interlock
  3. Send that assignment to the plant

- Extracting explicit interlocks requires finding the set:
  \[
  \{(y \leftarrow \phi) \in \Phi \mid y \in \mathcal{O} \text{ and } \text{Var}(\phi) \cap \mathcal{O} \neq \emptyset\}
  \]

- Extracting implicit interlocks requires verification techniques
  - Thus, rewriting logic to contain only implicit interlocks can make interlock exploitation harder
Inferring process structure

• Some processes tend to be more **event-driven** while others are more **logic-driven**.

• The latter is most common in manufacturing, traffic control, and sequence control applications.

• For logic-driven processes, extracting the main loop can be useful for a number of things:
  
  ‣ Determining where to hook malicious code
  ‣ Finding terminal states, especially those that depend only on outputs. (These are indicative of alarm conditions.)
• Data dependency graph for logic variables.
• Only the class of a variable is known (input, output, state, or timer).
• In this example (traffic light system):
  ‣ The main timing loop can be seen.
  ‣ o6 (alarm condition) is interlocked to o4 and o1 (conflicting green lights).
Summary

• The individual tasks needed for constructing dynamic malware payloads seem feasible.

• Arguably, the hardest and most expensive task is the collection of disassemblers and device databases.

• Plants can be forced to behave unsafely even if no device information is available.

• Malware authors can leverage existing program analysis techniques like dependency graphs to design dynamic payload mechanisms.

• We are looking for test cases!
Thanks!

smclaugh@cse.psu.edu