Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

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

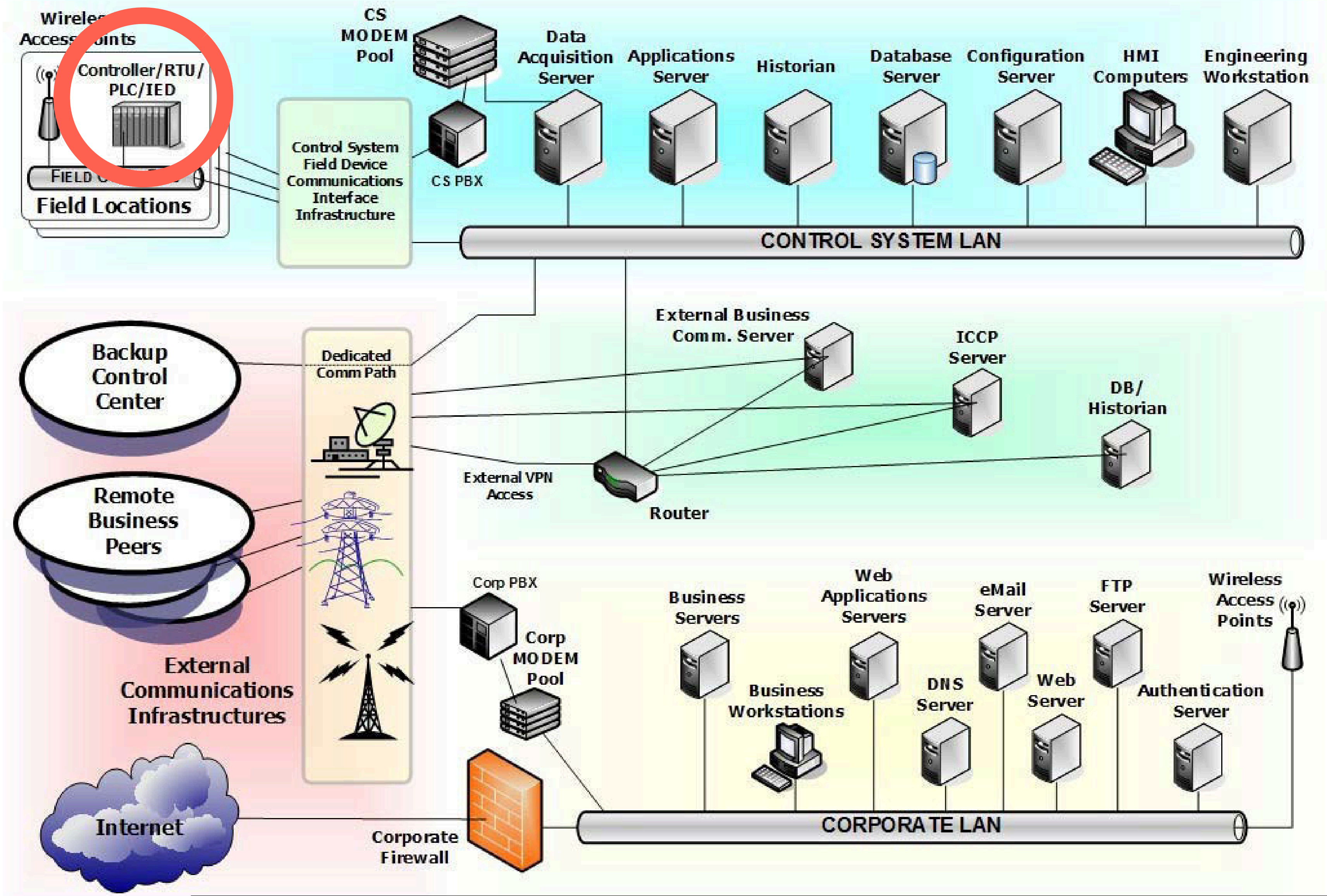


Image source: *Control Systems Cyber Security: Defense in Depth Strategies*. Idaho National Laboratory. 2006

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.

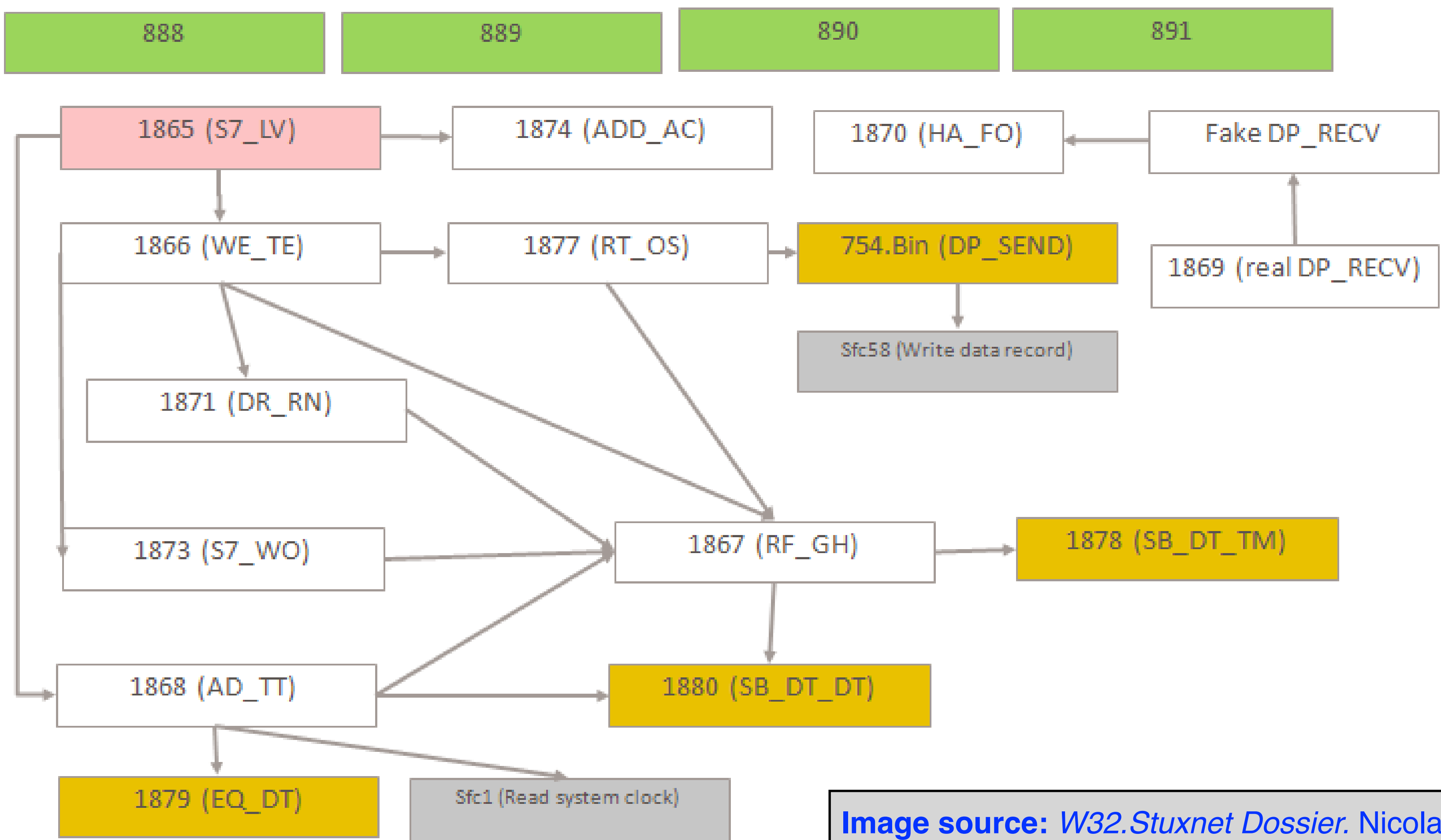


Image source: *W32.Stuxnet Dossier.* Nicolas Falliere, et al. 2010

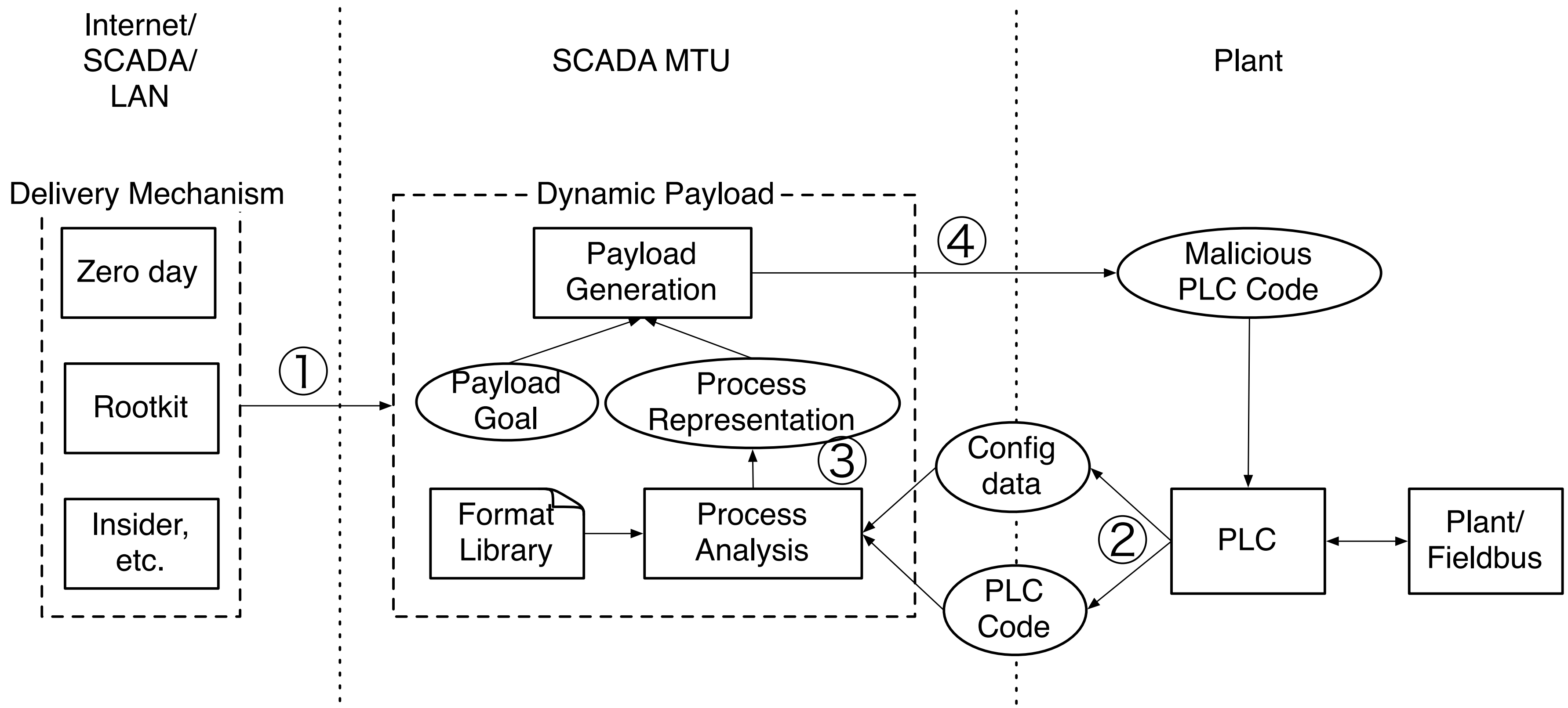
Wednesday, August 10, 2011

This talk

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.

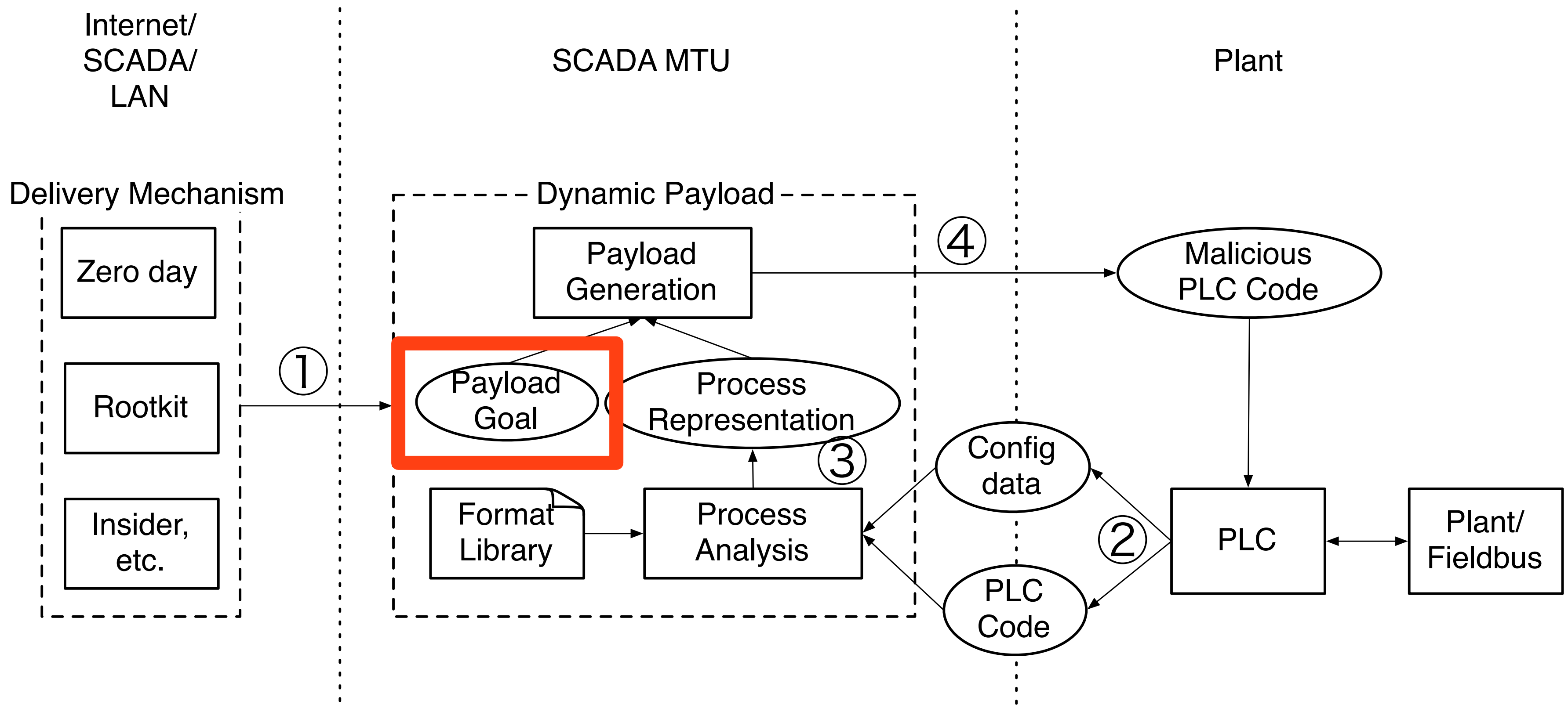


- This problem can be solved, as can all software engineering problems, through a breaking down into modular steps.

Wednesday, August 10, 2011

Code reuse

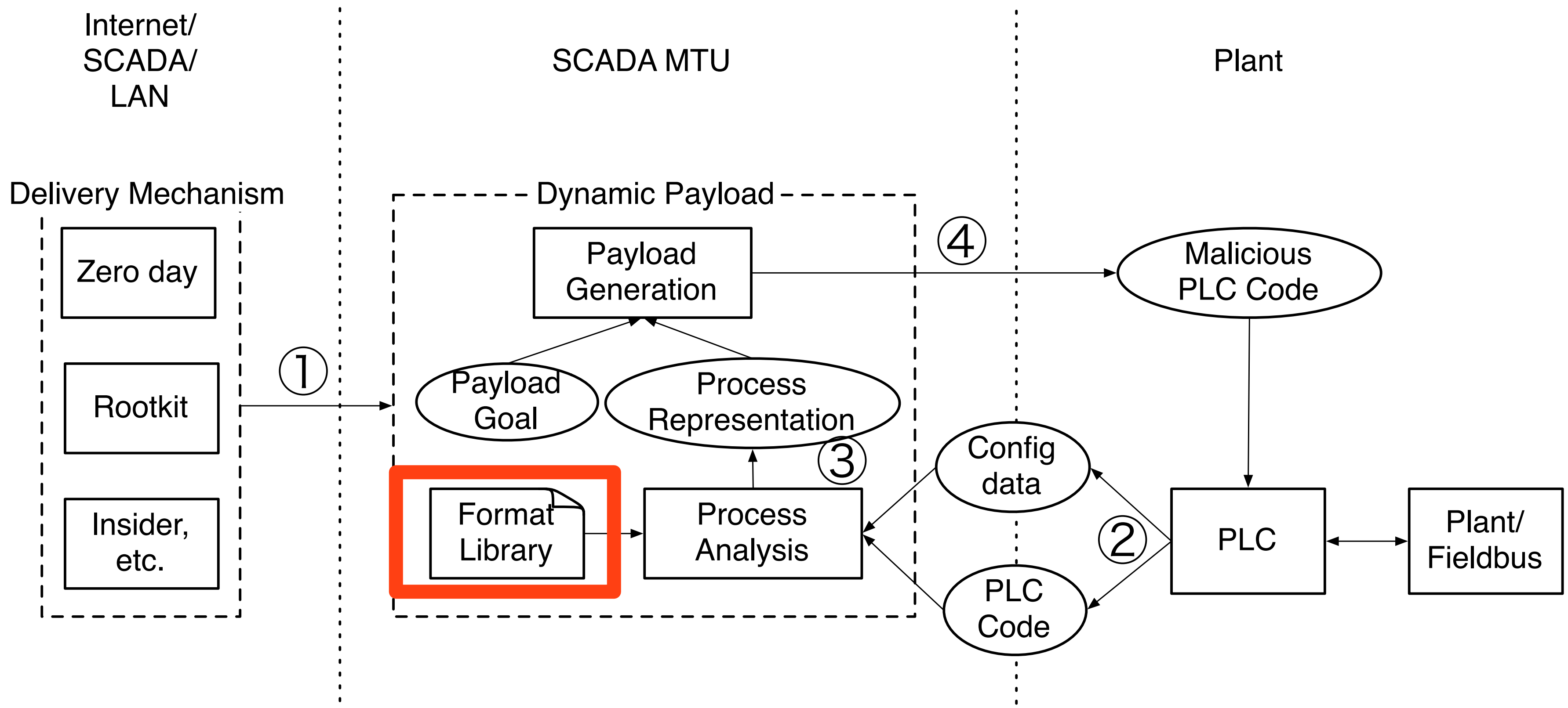
Ideally, the adversary need only specify the payload goal.



Wednesday, August 10, 2011

Code reuse

The format library contains platform-depended disassemblers and device IDs.



Wednesday, August 10, 2011

Logic programming

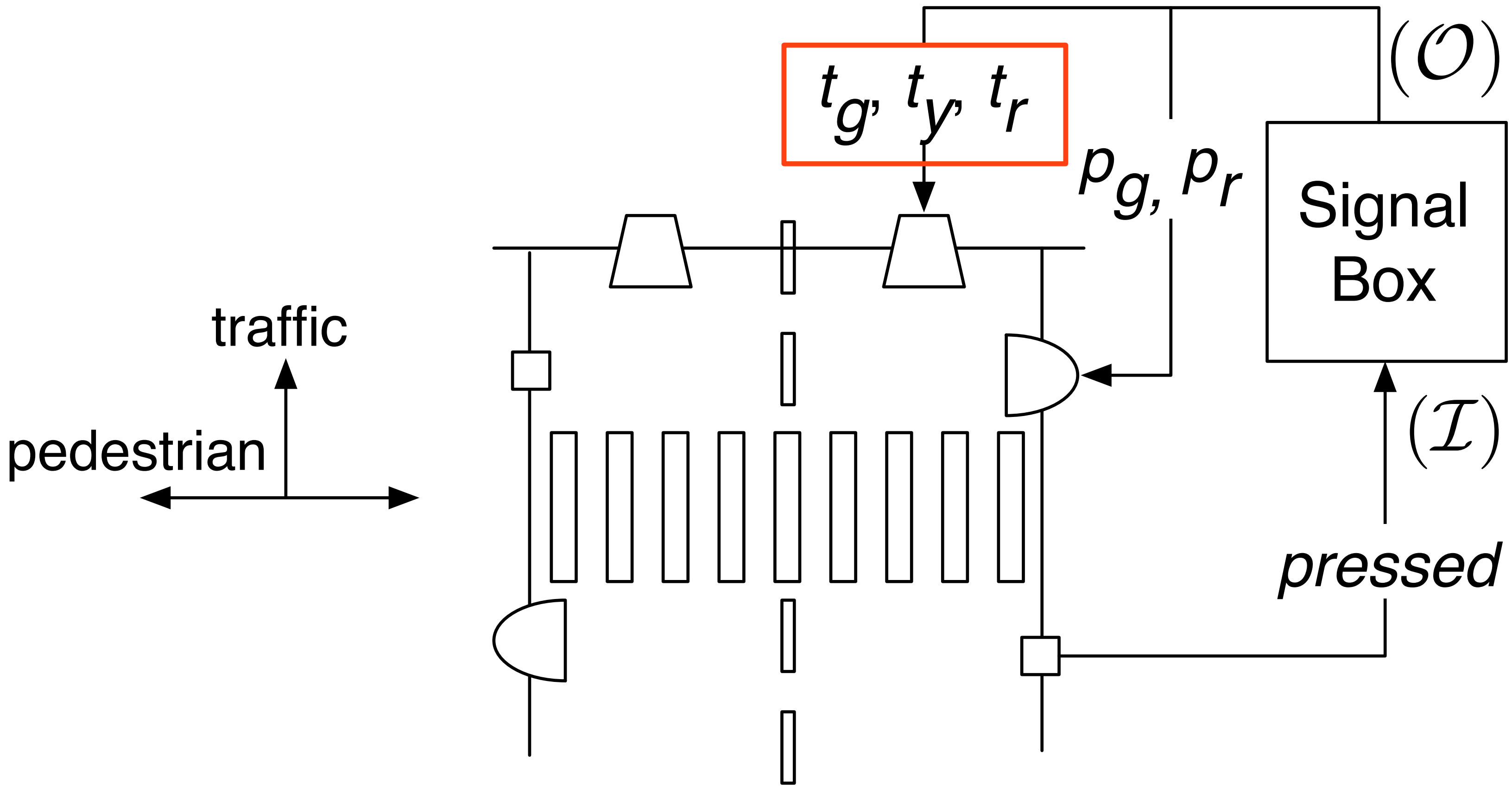
- Logic programs simulate Boolean circuits.
- A PLC program maps a set of input variables $\mathcal{I}$ to a set of output variables $\mathcal{O}$.
- Values for $\mathcal{I}$ are received from the sensors in the plant, and values in $\mathcal{O}$ are sent to the plant to manipulate devices.
- A set of internal state variables $\mathcal{C}$ and timer variables $\mathcal{T}$ are also available.
- A logic program is a set of expressions Φ s.t.

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

- Note that in practice, we can often differentiate the four types of variables.

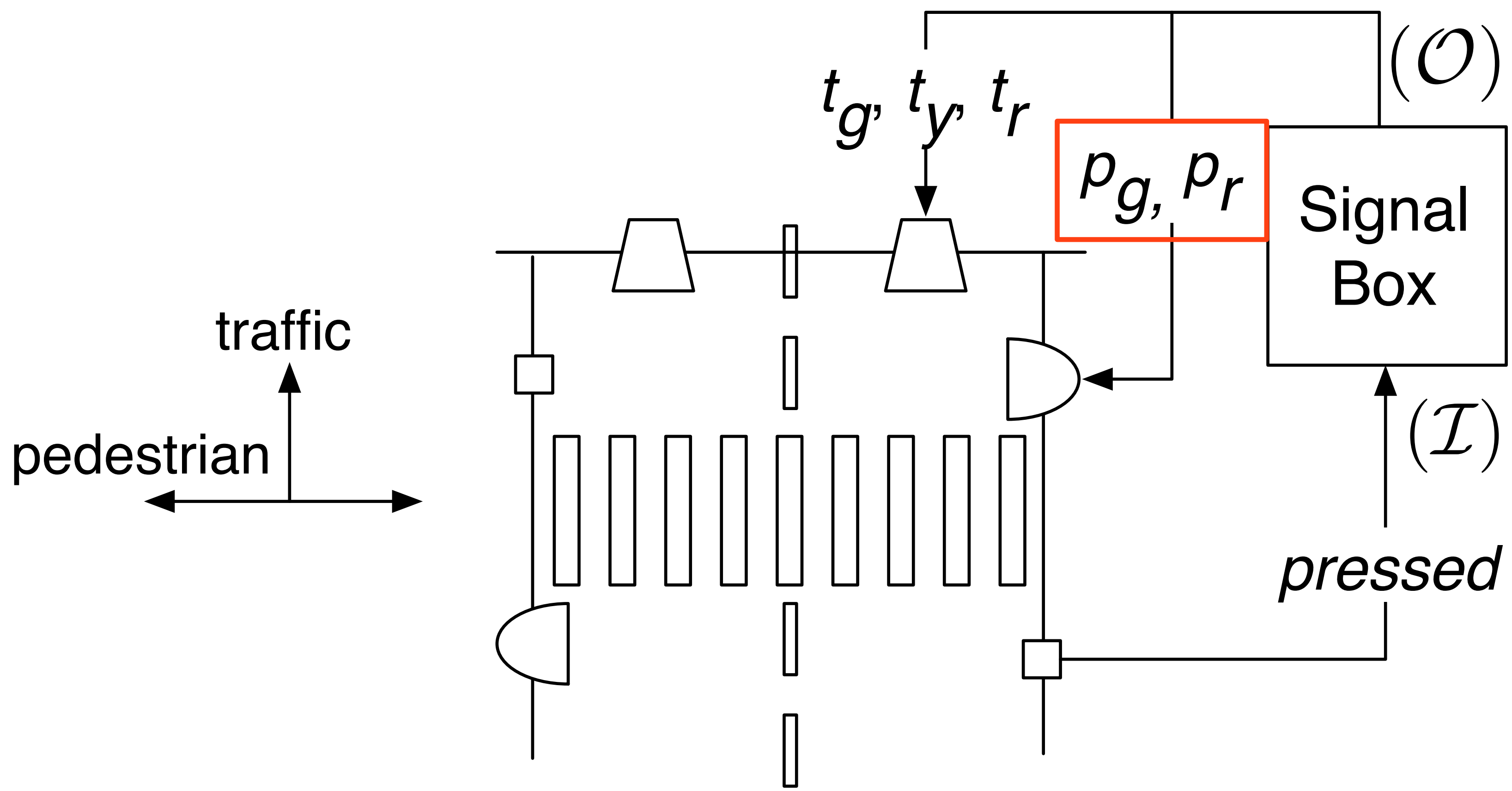
A pedestrian crossing

Traffic green, yellow, and red (output)



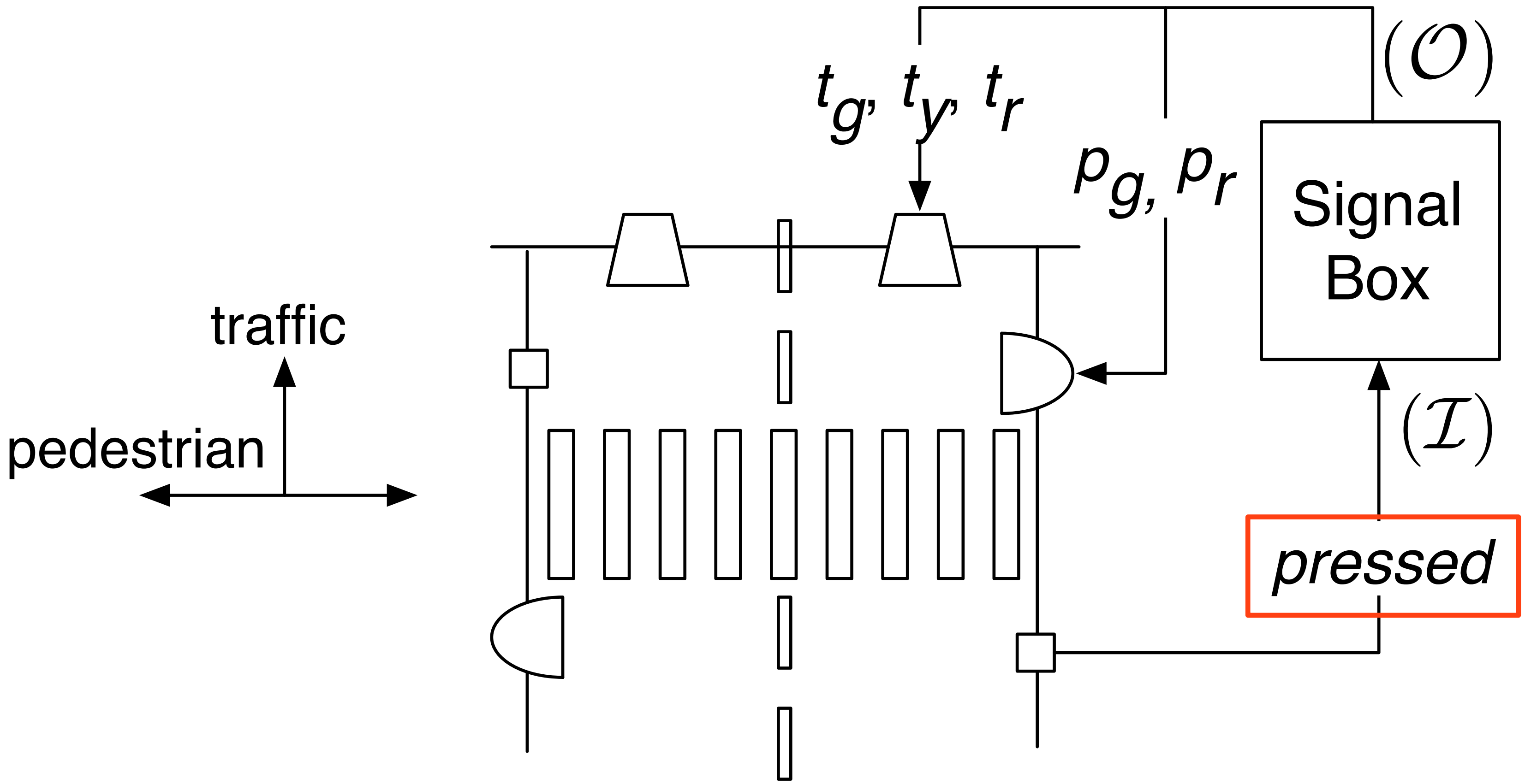
A pedestrian crossing

Pedestrian green and red (output)



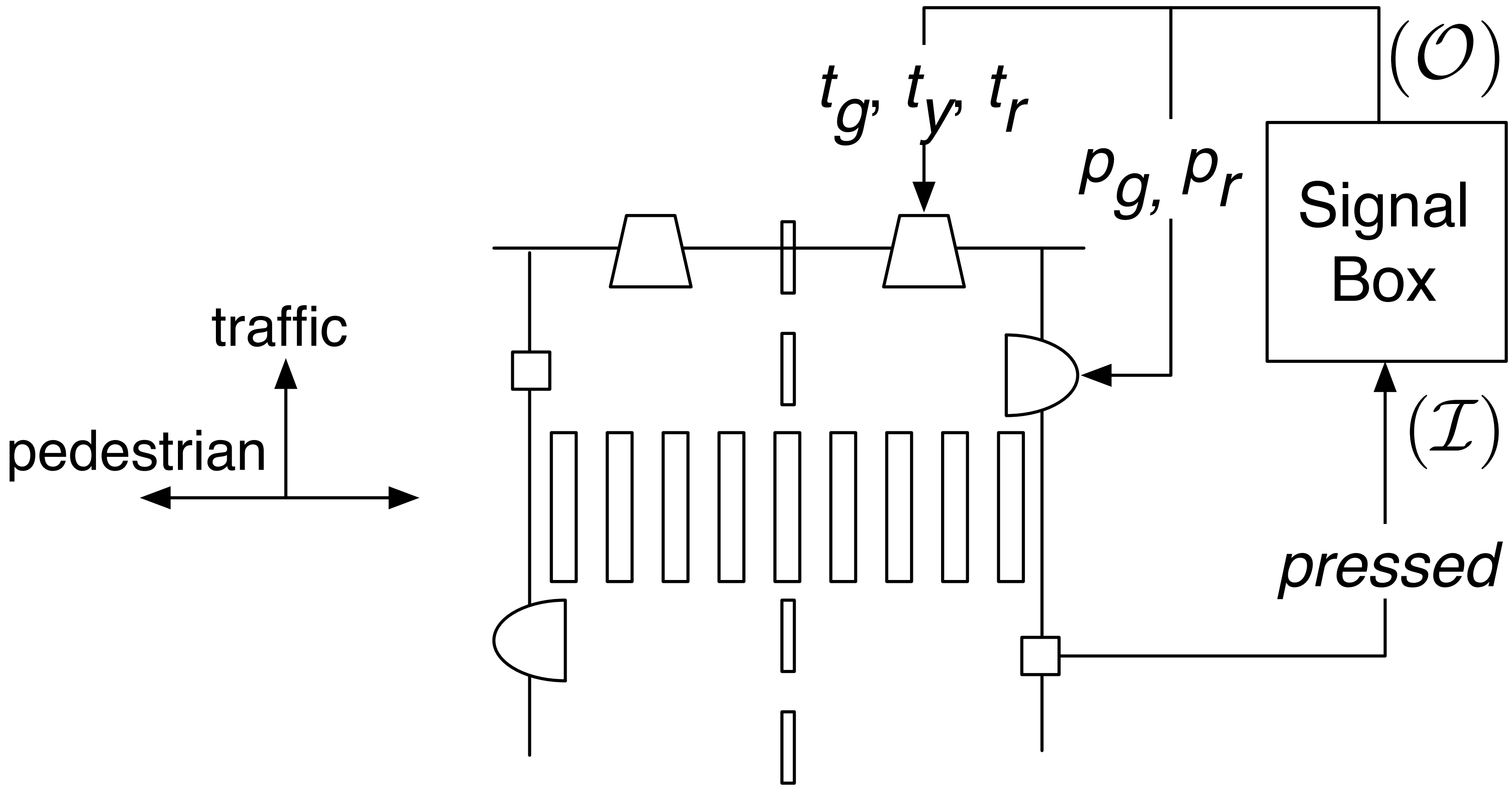
A pedestrian crossing

Request to cross button (input)

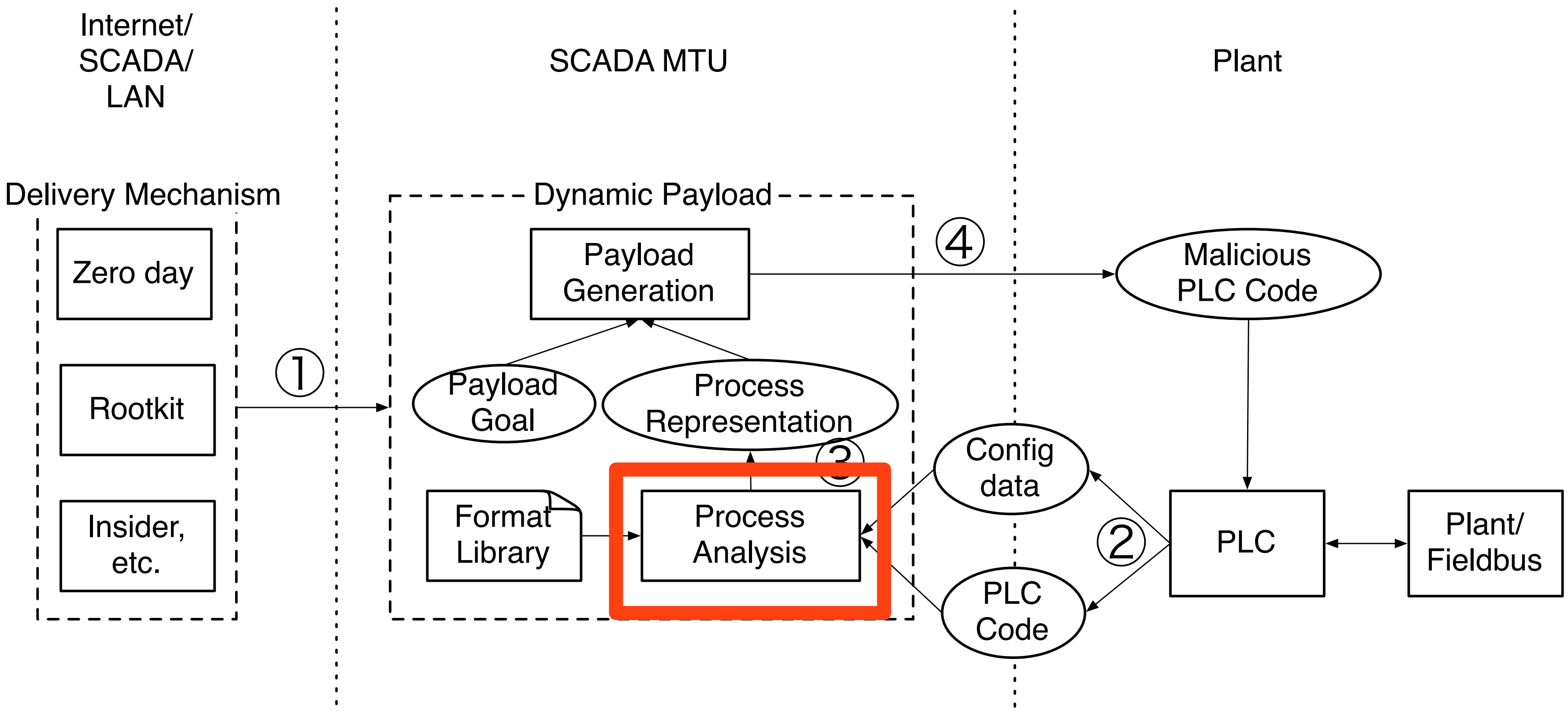


A pedestrian crossing

$$(y \leftarrow \phi) = (p_g \leftarrow pressed \wedge t_r), \quad \mathsf{Var}(\phi) = \{pressed, tr\}$$



Process analysis



Wednesday, August 10, 2011

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 Φ 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!

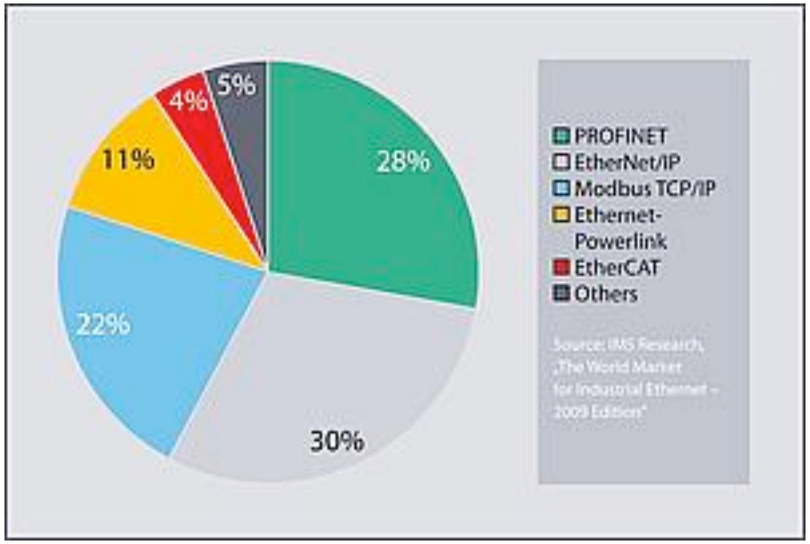
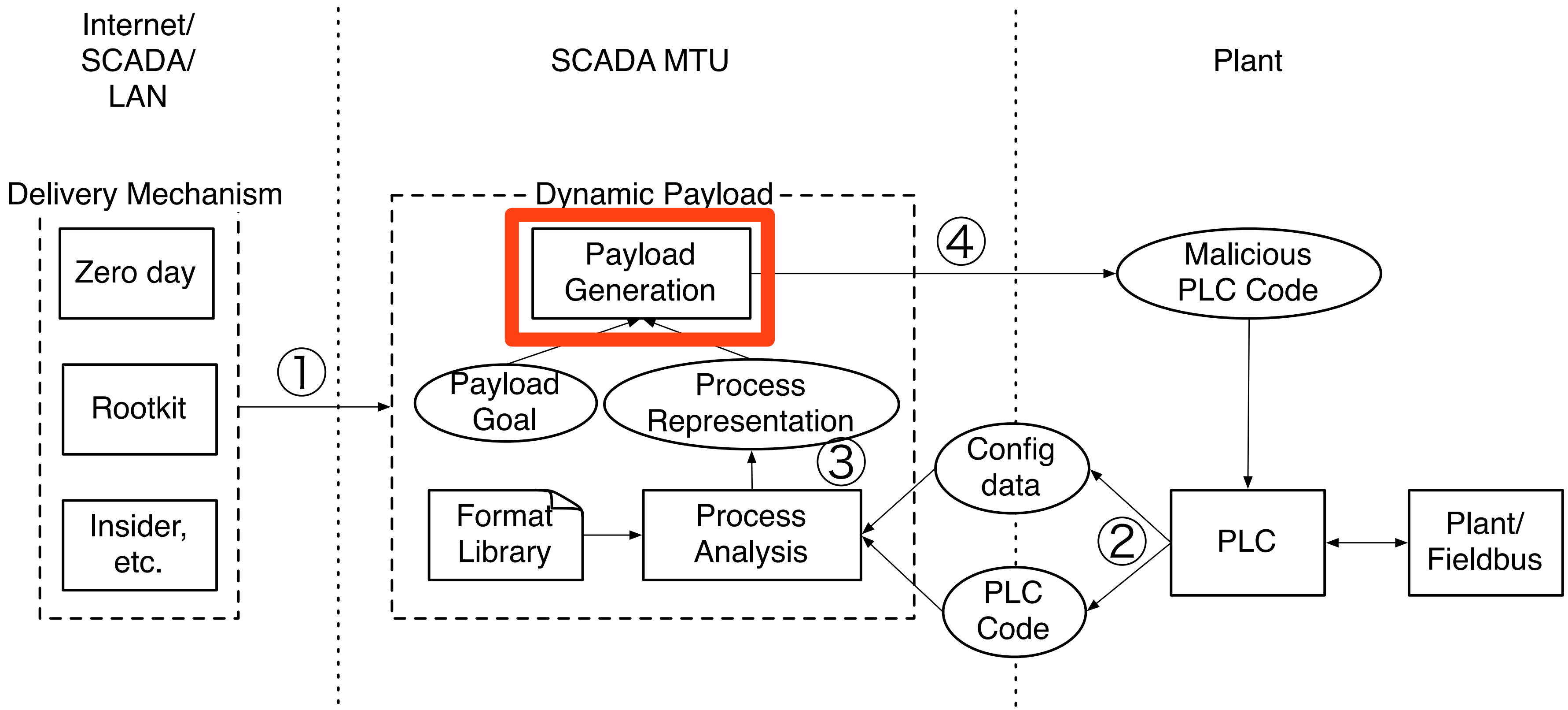


Image source: `profibus.com`

Payload generation



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:

$$\neg(p_g \wedge t_g)$$

  - 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:

$$\{(y \leftarrow \phi) \in \Phi \mid y \in \mathcal{O} \text{ and } \mathsf{Var}(\phi) \cap \mathcal{O} \neq \varnothing\}$$

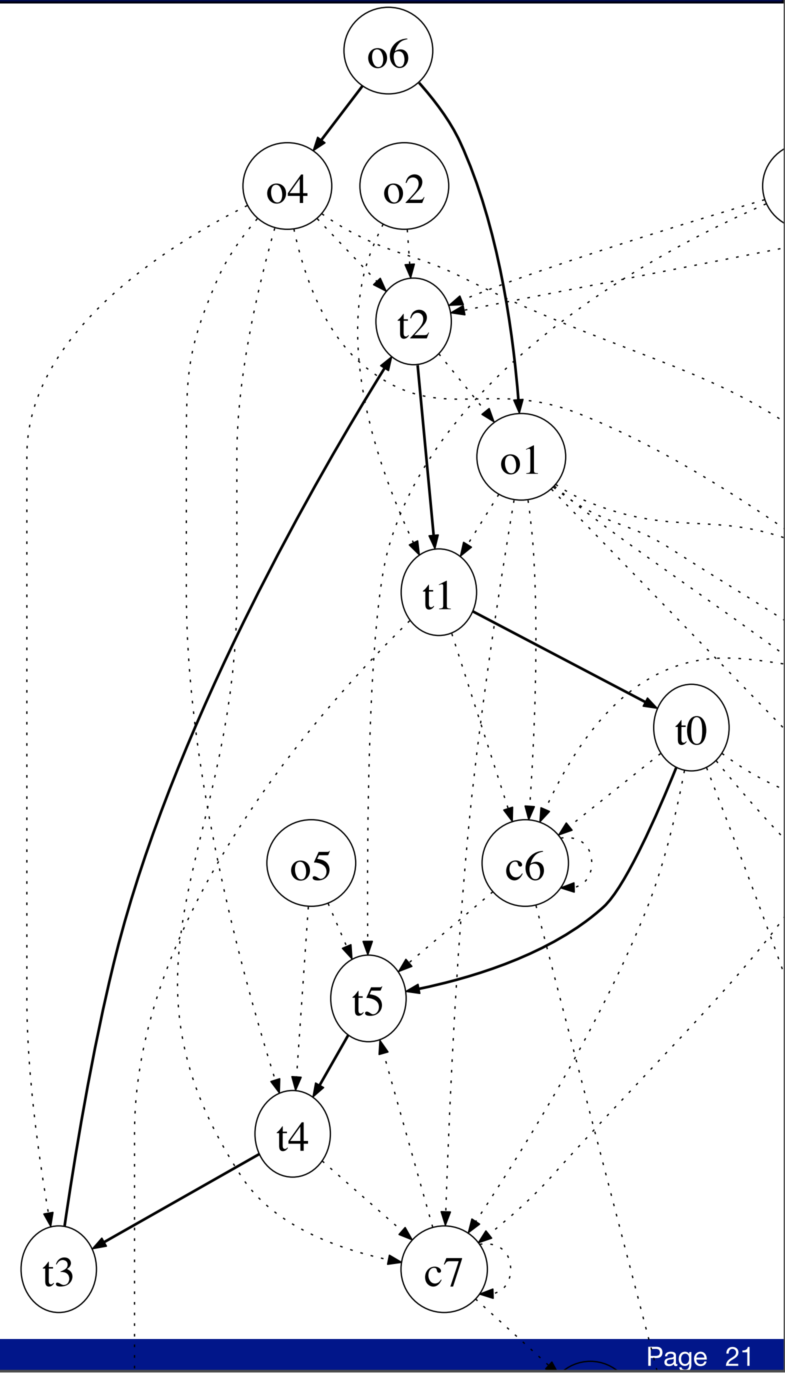
- Extracting implicit interlocks requires verification techniques
  - Thus, rewriting logic to contain only implicit interlocks can make interlock exploitation harder

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.)

Process dependency graph

- 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