Safety and Security Analysis in Cyber-Physical Systems using Metro-II

Purva Tendulkar

with Arvind Easwaran, Anupam Chattopadhyay

August 4, 2017

Contents

1 Introduction

Cyber-Physical Systems play an increasingly important role in critical infrastructure, government and everyday life. Automobiles, medical devices, building controls and the smart grid are examples of CPS. Each includes smart networked systems with embedded sensors, processors and actuators that sense and interact with the physical world and support real-time, guaranteed performance in safety-critical applications. Security considerations for these systems has also become very important today. The deployment of newer devices and software technologies has led to an increase in the number and types of attack surfaces. If an attacker has good knowledge about the system in use, he may exploit certain (zero-day) vulnerabilities which were compromised in the process of developing the system. Designers are ususally concerned with the working of the system without consideration of how easily it could be penetrated into as a result of the rapid changes and advancements being made to it. We try to emphasize the importance and complexity of safety-critical analysis and modeling before system-level deployment.

To design a secure system we need to understand its possible threats, more specifically we need to understand how the components of the architecture are compromised and used by an attacker in order to fulfill his objectives and how the attack proceeds through these components. While there already exist modeling frameworks for modeling attacks for standalone networks or hardware devices, we are trying to analyze and use frameworks that can capture functionality at all the layers of a system including software tasks, RTOS, and hardware components and model the flow through all these layers. We believe that this will lead to more detailed analysis on possible attack surfaces and can help us analyse all the possible threats to the system.

On these lines, I have studied Metropolis (Metro-II), a framework developed by U.C. Berkeley. Metro provides a Design Environment for Heterogeneous Systems which allows the user to define a system at all levels of abstraction. It was created with the intention of allowing developers to capture their designs and to provide proper separation of functionality/logic from the architecural set-up in order to analyse each; individually and together. I used this framework to first model some simple attacks as proposed in the paper (Easwaran et al., 2017). I then studied a specific attack (Stuxnet) in detail and used Metro to model this attack.

2 Acknowledgements

I'd like to thank my advisors Prof. Arvind Easwaran and Prof. Anupam Chattopadhyay for some excellent discussions and for keeping me motivated throughout the 12 weeks. I am grateful to have been able to make full use of the HESL lab and its resources and to have gotten the opportunity to be a part of the research environment at NTU. I am thankful to all the PhD and Masters students present in the lab for their help and support.

3 About Metropolis

The Metropolis metamodel can support not only functionality description and analysis, but also architecture specification and the mapping of functionality to architectural elements. Metropolis offers syntactic and semantic mechanisms to compactly store and communicate all relevant design information so that designers can use it to plug in the required algorithms for a given application domain or design flow. Different variations of the model can be explored by changing the parameters of architectural elements such as memory size, number of processors or designing different scheduling algorithms. The choice of technique or algorithm for analysis and synthesis of a particular design depends on the application domain and the design phase. For safety-critical applications, formal verification techniques, which require significant human skills for use on realistic designs, are needed. Details about the Metropolis Meta Model can be found here (Davare et al., 2013), (Balarin et al., 2003).

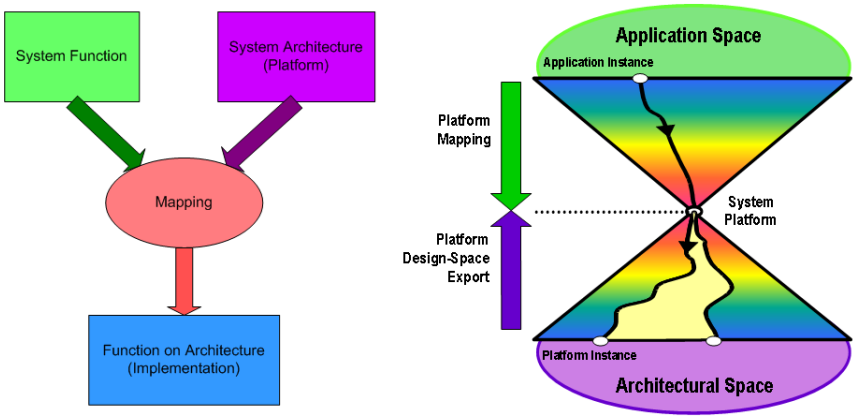

Figure 1: Metropolis Modeling

4 PIP Model

Metropolis modeling for a complete system involves 3 key parts - Functional modeling, Architectural modeling and Mapping of the functional model on the architectural elements. The details of this model are provided here (Zeng et al., n.d.). I will give a brief description of these 3 stages-

4.1 Function modeling

A systems function is the set of processes that concurrently take actions while communicating with one another. Each process is made up of a sequential thread. A process communicates through the ports defined in it. A port is specified with an interface, which declares a set of methods that the process can use through the port. The objects that implement port interfaces are called media. Any medium can be connected to a port if it implements the ports interface.

Thus Metropolis allows a separation between the computation and communication of processes. This separation is essential to facilitate the description of the objects to be reused for other designs. Once a network of processes has been established, we can use the metamodels semantics to precisely define the networks behavior as a set of executions. We can define a processs execution as a sequence of events, which are a programs entries or exits to some piece of code. A network execution is then defined as a sequence of event

vectors in which each vector has at most one event for each process to define the set of events that happen altogether. The metamodel can model nondeterministic behavior thus leading to more than one possible execution of the network.

Constraints, written in logic formulas, further restrict the set of executions defining the set of legal executions.

4.2 Architectural modeling

Two aspects distinguish architectures: the functionality they can implement and that implementations efficiency. To represent an implementations efficiency, we must model the cost of each service. This is done by first decomposing each service into a sequence of events, then annotating each event with a value representing the events cost. To decompose the services into sequences of events, networks of media and processes are used, just as in the functional model. These networks often correspond to the physical structures of implementation platforms, eg. Tasks, CPU, Bus, Memory.

The services that this architecture offers are the execute, read, and write methods that the Task processes implement. The thread function of a Task process repeatedly and nondeterministically executes one of the three methods. In this way, we model that the tasks can execute these methods in any order. The actual order will become fixed only after the designer finishes function-mapping to the architectureso that each task implements a particular process of the functional model. While a Task process offers its methods to the systems functional part, the process itself uses services that the CPU medium offers, which, in turn, uses the BUS mediums services. In this way, the system decomposes the top-level services that the tasks offer into sequences of events throughout the architecture

The metamodel includes the notion of using quantity to annotate individual events and using values to measure cost. Quantities can also be used to model shared resources. (eg. time is being modeled as a sequence of reads and writes). Metropolis provides standard libraries for managing common quantities, such as time.

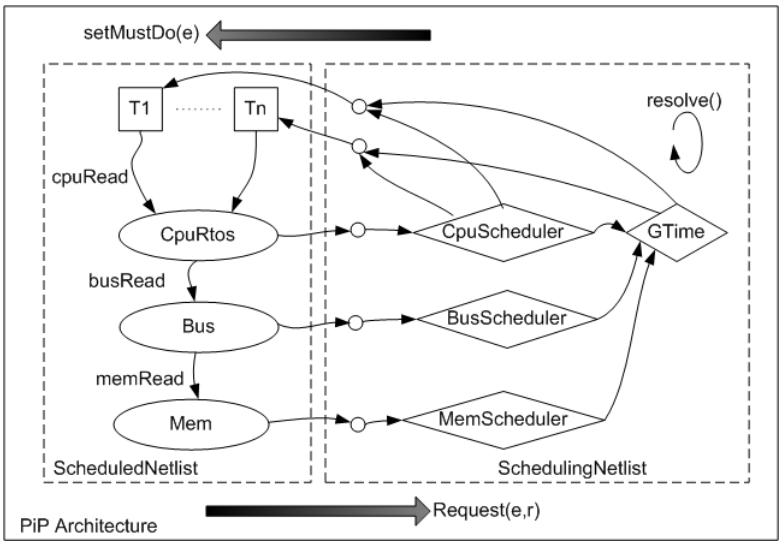

Figure 2: PIP Model Architecture

4.3 Mapping

Evaluating a particular implementations performance requires mapping a functional model to an architectural model. The metamodel can do this without modifying the functional and architectural networks with the help of Netlists. Each process in the functional model is mapped to a Task in the architectural model.

5 Modeling a simple attack

This attack has been mentioned in (Easwaran et al., 2017) paper.

5.1 Without Attack

The functional network consists of 4 processes - 2 source processes, 1 scheduler process and 1 sink process. This application is modeled in Metropolis as a process network consisting of a number of processes and channels. The 2 source processes send their requests to the scheduler through independent channels. The scheduler process waits for incomming requests across both these channels. When it gets the first request, it will grant it permission by sending a reply to that process along another channel. It will not receive any other requests until the owner process sends a reply to it asking for release. Once it receives this reply, it will then accept the request from the other source and follow the same procedure by sending it a grant and wait for release. After both processes have released their control of the CPU, the scheduler sends a message to the sink process and terminates.

This functional model runs on an architectural model composed of a CPU/RTOS component, a bus and a memory. The tasks in the architectural model are also called mapping processes, which serve as the interface between the architectural components and the functional model during the mapping stage. The tasks nondeterministically model all the possible programs that could be executed on the architecture. The software tasks are connected to the CPU/RTOS and use the services that those components offer for computation or communication. The CPU/RTOS is connected to the Bus. This component is a multi-master multi-slave communication device that allows the CPU/RTOS to communicate with the Memory. The Memory is a slave device connected to the Bus.

The CPU/RTOS component is shared among several tasks. Each of these could require a service. When more than one request is issued to the CPU/RTOS, arbitration is needed. We can think of the CPU as a device that must be distributed among the tasks. Each time there is a request from a task, the request is passed to the CpuScheduler quantity manager which decides the task that will be the owner of the CPU/RTOS. The same procedure occurs when more than one master asks for ownership of the Bus or Memory. In this case, the BusScheduler or MemScheduler quantity manager will decide the winner. There is another quantity manager called GTime which annotates instances of events with the Global Time physical quantity.

5.2 With Attack

In a real-time CPS, a possible attack could be an event that results in a missed deadline for a real-time task. In the above system, if the first source process does not release its control over the CPU for a long time, the second source process could get delayed resulting in a deadline miss! This non-relase of CPU by the first source process could be due to a number of reasons like malicious software logic embedded in the code of this process, or the power supply (or clock) to a processor getting disturbed. I was able to model both

these scenarios by-

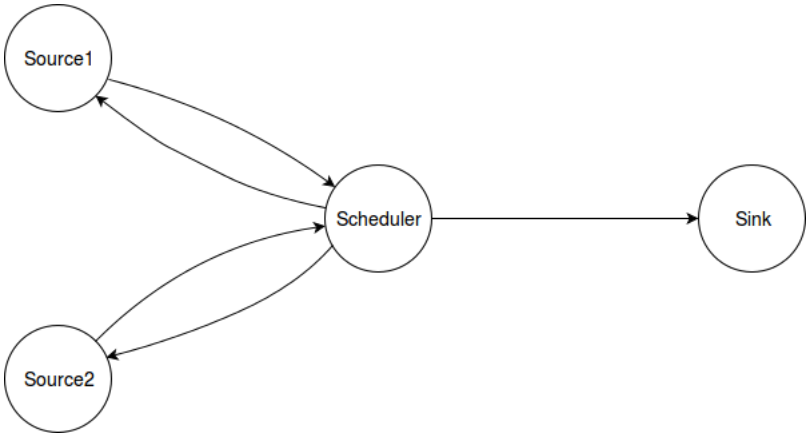

Figure 3: Functional model for a simple attack in Metropolis

1. Introducing malicious logic in the process which makes the process do a lot of time-consuming activities (like memory reads and writes) before releasing control
2. Changing the system clock cycle time period the moment control goes to this process

6 About Stuxnet

6.1 Overview

Stuxnet is a malicious computer worm which targetted specific Siemens PLCs which controlled Iran's nuclear power-plant. It used a number of Windows system vulnerabilites and then presented itself as physical world output by making the centrifuges spin at extreme rates, thus attempting to disrupt plant operation. It was also very stealthy and was designed so that the plant operators wouldn't notice anything going wrong on their monitors. The details about Stuxnet can be found here (Falliere et al., 2011). A brief description of the attack can be found here (Mueller & Yadegari, 2012). We limit our discussion to the sequence of events occuring after Stuxnet has reached the target computer. We will not be dealing with the vulnerabilities exploited in the process of traveling through the network and updating itself to the latest version via HTTP requests, as these were simply means of increasing its chances of reaching the target system, and did not directly affect any of the intermediate systems.

6.2 Architecture of the Industrial site

The whole facility architecture is composed of two main security zones, the Business Corporate Network and the SCADA system. The details of this set-up are mentioned in (Kriaa et al., 2012) paper.

The Business Corporate Network hosts the Enterprise usual Information System. It comprises servers and workstations that enable classical daily applications (emails, reporting, accountability...), the Enterprise Resource Planning (ERP) system, etc. This network can exchange data with external networks connected to the Internet through a "demilitarized zone".

The SCADA system includes a Process Control Network and a Control System Network. The Process Control Network consists of WinCC and PCS7 clients and servers which are connected to PLCs and enable communication with them. WinCC machines provide

HMI client/server systems used to monitor the industrial process and visualize messages and real-time data. PCS7 machines include basic data collection functions for project data, process values, archives, alarms and messages. PCS7 servers provide all process data and connect PLCs to the ProcessControl Network. The control system network includes WinCC/PCS7 servers and PLCs. PLCs send control signals via a Process Field Bus (Profibus) to speed regulators that control the rotation of motors. For security reasons, the SCADA system is isolated by an air gap so that no network connection is possible between the two security zones.

The setup inside the SCADA control system can be expressed in terms of Metropolis processes as follows-

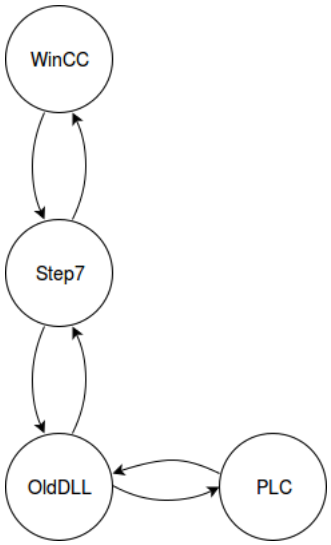

Figure 4: SCADA system before attack

- WinCC sends read/write request to Step7 to request some data from the PLC/Step7 program.
- Step7 can either return data which it holds with itself or request data from the PLC by sending a request to the DLL.
- OldDLL gets this request and sends it to the PLC to read/write from it. It will then return the appropriate data back to the WinCC process via the Step7 program.

6.3 Propagation

Due to the presence of the air-gap, Stuxnet needed to have been introduced to the SCADA control system via a removable drive (eg. a pendrive). Once the pendrive was inserted, it spread via an autorun.inf file. An autorun.inf file is a configuration file placed on removable drives that instructs Windows to automatically execute a file on the removable drive when the drive is inserted. Stuxnet uses a single executable file which also contains a correctly formatted autorun.inf data section at the end. When autorun.inf files are parsed by the Windows OS, the parsing is quite forgiving, meaning that any characters that are not understood as legitimate autorun commands are skipped. During parsing of the autorun.inf file all of the MZ file will be ignored until the legitimate autorun commands that are appended at the end of the file are encountered.

It then exploits a Windows zero-day vulnerability (we limit our discussion to the Keyboard Layout vulnerability) to exploit its priviledges and gain root access (operate in kernel mode). The vulnerability is caused when the Windows kernel-mode drivers do not properly index a table of function pointers when loading a keyboard layout from disk. An attacker who successfully exploited this vulnerability could run arbitrary code in kernel mode. The attacker could then install programs; view, change, or delete data; or create new accounts with full user rights.

Stuxnet installs two kernel-mode drivers, one for running Stuxnet after reboot and the other as a rootkit to hide its files. It is able to perform all these actions by making use of 2 stolen digital certificates (JMicron, RealTek). After gaining root access, Stuxnet will then move on to the main attack phase.

6.4 Attack

Stuxnet will look for Step7 project files to infect and then drops its main malicious DLL file onto the system. This main file performs the following-

- Spends time (13 days) doing nothing on the system but monitor the requests and responses being sent; thus gathering valid data of the regular system functioning.
- Periodcially sends commands to the PLC to make it spin the centrifuges at extreme rates.
- Intercepts calls made from the WinCC client (via Step7 programs) to the Original DLL and sends false data (gathered initially) back to the WinCC program to mask the fact that the system has been compromised.

7 Metro Model for Stuxnet

7.1 Model

I have modeled the functional network consisting of 2 main phases - Propagation and Attack

The Propagation phase consists of 4 main processes-

- **Pendrive** : Represents the pendrive process running on the target Windows machine.
- **Layout** : Represents the program running on the Windows machine that loads the Keyboard layout in memory.
- **Key** : Represents the action where a user presses a key on the keyboard.
- **Overflow** : Represents the process which runs once a key on the keyboard has been pressed. Here, this process exploits the Keyboard vulnerability by causing an overflow while accessing a table index in memory.

The Attack phase consists of 7 main processes-

- **WinCC** : Represents the user program communicating with the PLC. This communication may be for one of the 2 reasons -
  - To request certain values from the PLC or Step7 program to check the functioning of the PLC (reading from its memory)

– To send signals to the PLC to make it behave in a certain way (writing to its memory)

- **Step7** : Represents the main program on the SCADA Control Computer which interacts with the PLC via the DLL. This element was compromised by Stuxnet.
- **Step7Periodic** : Represents the compromised counterpart of Step7 component. Contains the malicious logic of Stuxnet which periodically sends malicious data to the compromised DLL.
- **NewDLL** : Represents the malicious DLL which intercepts requests which should actually have been received by the OldDLL and subsequently forwarded to PLC. This element replaces the original OldDLL element in the Stuxnet attack.
- **NewDLLPeriodic** : Represents the counterpart of the malicious DLL which handles malicious incoming requests from Step7Periodic. Whenever Step7Periodic sends a request to attack the PLC operation, NewDLLPeriodic sets a variable in the medium to indicate to the malicious DLL (NewDLL) that request has been made.
- **OldDLL** : Represents the original DLL (s7otbxdx.dll) which Step7 uses to communicate with the PLC. It is being replaced by NewDLL in the Stuxnet attack and is being referenced by NewDLL for some non-critical functionality.
- **PLC** : An abstraction which represents the Controller which directly controls the centrifuges. The attacker cannot directly attack and compromise the PLC functionality and its logic remains untouched in the attack.

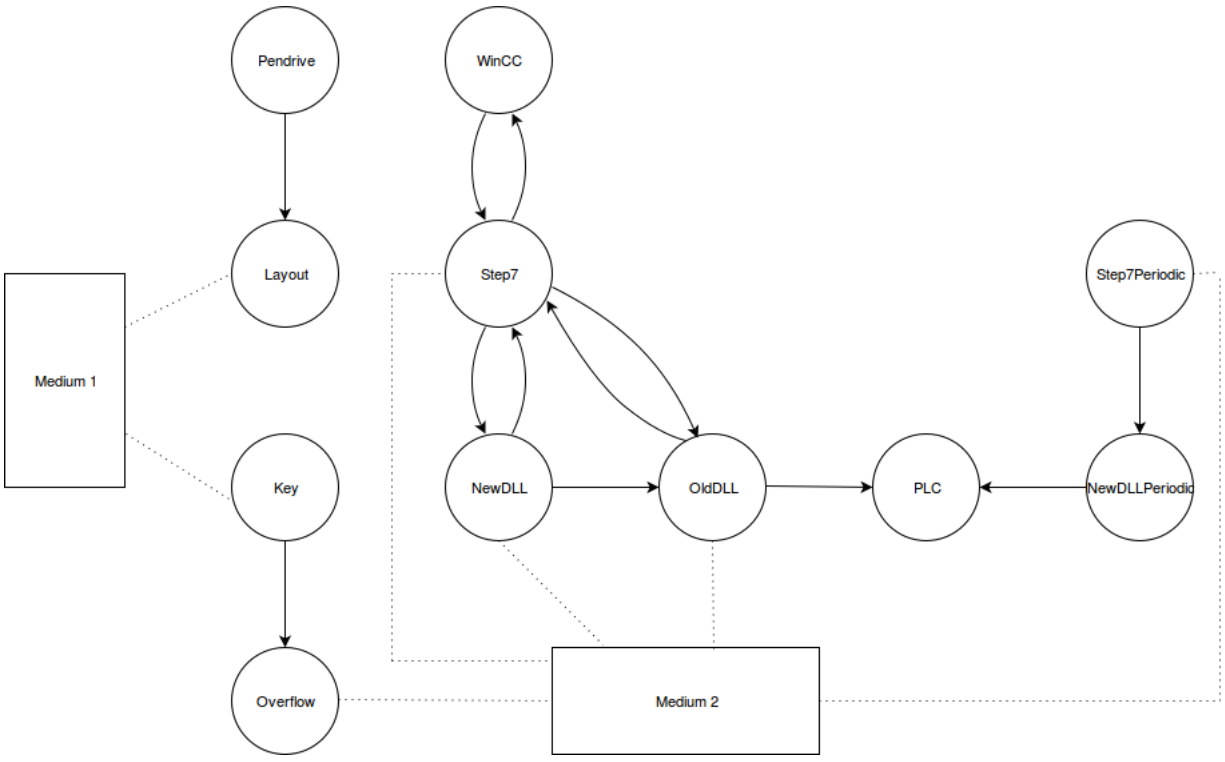

Figure 5: Compromised SCADA system during Stuxnet attack

The dotted lines indicate that processes are able to access some data from the media. This may be used for indicating that a particular event has occurred.

7.2 Execution

7.2.1 Propagation

Process from the pendrive (Stuxnet) automatically starts executing on the target computer using the autorun.inf. It contains its own Keyboard layout file which it loads into a specific location in memory. It then waits for the user to press any key on the keyboard which triggers a function. This function (xxxKENLSProcs) contains the actual vulnerability which is being exploited by Stuxnet. The function points to a table in memory which contains only 3 entries but the pendrive code has inserted appropriate data in the memory so that this function invokes index 5 in the table. The vulnerability lies in the fact that no check was being made on the index in this function. This transfers control to a region in user memory which Stuxnet had access to and into which it had already planted malicious code. Thus, Stuxnet was able to exploit the vulnerability and essentially run its own code in the context of the kernel (that is, with kernel-level privileges). A brief description of how Stuxnet exploited the Keyboard Layout vulnerability can be found here (Helan, 2011).

This is modeled in Metropolis as follows-

- Pendrive process does some memory writes and then sends a request to Layout process to load the new keyboard layout.
- At some instant of time, Key process independently sends a request to Overflow process, indicating that a key has been pressed and that the appropriate function take charge - which leads to escalated privileges.

Once Stuxnet has root-level access, it looks for Step7 programs, injects itself into them and drops its malicious DLL which takes control in the attack phase.

7.2.2 Attack

Following is the attack sequence once BadDLL, and consequently BadDLLPeriodic, have been introduced into the system.

- WinCC sends read/write request to Step7 to request some data from the PLC/Step7 program.
- Step7 can either return data which it holds with itself or request data from the PLC by sending a request to the DLL.
- If NewDLL gets this request, it has 3 options based on the type of data being requested-
  - If non-critical data is being requested, it is always provided
  - If data which is always critical is being requested, NewDLL handles it by sending false data which it has collected
  - If data which is critical only when malicious operation is running is being requested, following cases can occur-
    - If NewDLL has already realised, before incomming request alert from NewDLLPeriodic, that malicious logic is running, it simply sends back false data without consulting OldDLL or PLC. WinCC has no way of knowing whether PLC has actually been accessed or not and assumes that the data is valid real-time data.

    - If NewDLL has already sent request to OldDLL and then receieves malicious request alert from NewDLLPeriodic, it cannot stop request that has already been sent. But before it sends reply back to Step7 it checks if malicious operation is in progress and if it was, it does not send correct data, but intentionally sends the wrong data.

- Step7Periodic waits for a certain amount of time. During this time, Stuxnet does nothing but simply monitor the system listening to the values being read/written. This took place for 13 days.
- After this time, it sends requests periodically across its channel to NewDLLPeriodic.
- NewDLLPeriodic simply sets a variable in the medium indicating that the malicious operation is in progress and forwards the requests to the PLC whenever it gets a request across its channel.
- After some fixed time, Step7Periodic sends another request which travels in the similar fashion. The first request is to write malicious data to spin centrifuges for a fixed time duration at some undesirable rate. The next request is issued to reset the variables and resume regular operation.
- Then after some time (27 days), again a request is made to spin the centrifuges for a fixed time duration at an undesirable rate and then again reset it. This process continues every 27 days till some fixed date in June, 2012.

8 Conclusion

It is possible to make use of heterogenous modeling tools like Metro to model Cyber-Physical systems at different levels of abstraction. However, it is very complex to analyse the different system-level vulnerabilities without detailed prior knowledge of the system being analysed. Attacks, such as Stuxnet, can only be modeled when the proagation and attack path is already known. However, analysing the possible vulnerabilities that could lead to major compromise in an already developed system is very hard to do. Thus, it is important that security analysis be performed at the initial stages of deployment and that newer techniques are explored to perform security analysis for already existing Cyber-Physical Systems.

References

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., & Sangiovanni-Vincentelli, A. (2003). Metropolis: An integrated electronic system design environment. *Computer*, *36*(4), 45–52.

Davare, A., Densmore, D., Guo, L., Passerone, R., Sangiovanni-Vincentelli, A. L., Simalatsar, A., & Zhu, Q. (2013). metro ii: A design environment for cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, *12*(1s), 49.

Easwaran, A., Chattopadhyay, A., & Bhasin, S. (2017). A systematic security analysis of real-time cyber-physical systems. In *Design automation conference (asp-dac), 2017 22nd asia and south pacific* (pp. 206–213).

Falliere, N., Murchu, L. O., & Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, *5*(6).

Helan, D. (2011). Stuxnet: Analysis, myths and realities. *Actu Secu*.

Kriaa, S., Bouissou, M., & Pièvre-Cambacédès, L. (2012). Modeling the stuxnet attack with bdmp: Towards more formal risk assessments. In *Risk and security of internet and systems (crisis), 2012 7th international conference on* (pp. 1–8).

Mueller, P., & Yadegari, B. (2012). The stuxnet worm. *Département des sciences de linformatique, Université de lArizona, http://www. cs. arizona. edu/˜ collberg/Teaching/466-566/2012/Resources/presentations/2012/topic9-final/report. pdf*.

Zeng, H., Shah, V., Densmore, D., & Davare, A. (n.d.). *Simple case study in metropolis* (Tech. Rep.).