# A Framework for the Simulation of Attack using Model Checking

Ebot Ebot Enaw
University of Yaounde I
National Advanced School of Engineering

Djoursoubo Pagou Prosper
University of Yaounde I
National Advanced School of Engineering

## ABSTRACT
Over the past couple of years, the number of cyber-attacks and data breaches have considerably increased and so have the damages they cause, making cyber risk one of the primary concerns for top managers and world leaders around the world.

Public and private organizations are therefore obliged to deploy appropriate security solutions in a bid to protect their assets against threats over time.

However the complexity of information systems coupled with the interconnected nature of assets complicate efforts to identify the loopholes in an information system especially given the dynamism of cybersecurity where new vulnerabilities are discovered around the world daily.

In an effort to provide IT administrators with a rapid and reliable way of detecting loopholes, the paper proposes a framework that leverages formal verification concepts to provide an abstract model of an information system with specific properties aimed at verifying the security of assets.

The paper is structured as follows: section 1 introduces the article, section 2 presents some research papers related to this paper's topic, section 3 states the problem, section 4 presents the paper's contribution to research, and section 5 presents the proposed framework

## Keywords
Risk, vulnerability, attack surface.

## 1. INTRODUCTION
The diversity of assets and their interconnections within an information system complicates the identification of attack scenario as well as the application of appropriate security control aimed at preventing cyberattacks.

Therefore a method for automatically and formally detecting loopholes inherent in information systems is highly needed;

It is worth mentioning that due to the complexity of critical software, formal methods for specification and verification of models of software were developed to assist with the formal specification and verification of properties that software are supposed to satisfy.

In this vein, this paper proposes an approach inspired by formal specification and verification of software properties to develop an abstract model of an information system as well as the properties that the information system should satisfy in an effort to guarantee the security of its assets.

## 2. RELATED WORK
[1] Firstly, introduces the concept of software model checking which differs from traditional model checking in the sense that traditional model checking requires a manually written model of an application while software model checking works directly on the implementation of the software written in full-fledged programming language. Software model checking has two main components including abstraction that consists of automatically extracting the model of the application before verification and adaptation which consists of adapting model checking to a form of systematic testing that is applicable to industrial software. The latter technique was developed from two main perspectives namely systematic testing of concurrent software and systematic testing of sequential software. Concerning the systematic testing of concurrent software, the paper presents several methods namely software model checking using dynamic semantics, systematic testing with a run-time scheduler and also an approach of systematic testing for multithreaded application named DPOR (Dynamic Partial Order Reduction). Regarding the systematic testing of sequential software, the paper presents the static test generation, the dynamic test generation and the systematic dynamic test generation.

[8] Presented timed automata as a formalism for model checking real-time systems. It described in detail concepts related to timed automata and the process of verification of reachability inherent in them as well as concepts related to linear timed temporal logic and branching timed temporal logic. It then presented some extensions aimed at improving timed automata namely weighted timed automata which consists mainly of a timed automata added to an observer variable and Timed Games.

[7] Surveyed application of model checking to security especially in the domain of protocol verification where it is used to verify the secrecy and weak aliveness properties. It presented the *Dolev* and *Yao* model which has been one of the prominent tools to model the behavior of malicious actor and asserted that this tool is not efficient in real-world scenario. It then presented some alternatives such as the combination of symbolic and computational model and probabilistic model checking.

[4] Firstly, presented the similarities and differences between model checking and data flow analysis in terms of program representation, property representation and analysis algorithm and demonstrated that data flow analysis is efficient but not very precise while model checking is very precise but poses performance problems. It then proposed a method called configurable program analysis that consists of combining model checking and data flow analysis in an effort to improve precision and efficiency. It also presented some examples of an approach that combines model checking and data flow analysis such as Predicate analysis + constant propagation, Predicate analysis+ explicit-heap analysis and Predicate analysis+ observer automata. Finally, it demonstrated through some examples that the combination approach is very efficient and precise.

[6] Highlights the fact that when verifying some important systems like hypervisors, some low level features like memory management and cache are often left aside. It then proposes a formal approach for the verification of integrity-preserving countermeasures against cache storage side channel. It subsequently identifies conditions that must be met by a security mechanism to neutralize the attack vector and verified correctness of some of the existing techniques to counter both (instruction- and data-cache) integrity attacks. These conditions were later translated into formal theorems that can be applied to various hardware/software platforms.

[3] focuses on attacks exploiting human vulnerabilities or weaknesses. It then presents some of these attacks and proposes an approach to describe the behavior of users through folk models. It then uses formal methods to describe these behavior so as to verify whether they pose some risks to an information system. It latter applied the proposed approach on two case studies which revealed that this approach is effective in the sense that it helps uncover inherent vulnerabilities in IT policy which can be exploited through human weaknesses.

[5] proposes a formal approach to verify the safety of critical infrastructure while taking into consideration its dynamic environment. Firstly, it proposes the Dynamic Parametrized Architectures (DPAs), which allows for the modeling of components of the infrastructure. It then proposed a way to automatically translate the DPA and the properties into an array-based transition system and verified the properties with Model Checker Modulo Theories (MCMT).

# 3. RESEARCH PROBLEM

With the advent of cybecriminality, besides optimizing the efficiency of processes and activities, ICTs also represent a threat to them. In order to assess cyber threats that target information systems, IT departments usually conduct vulnerability scan on their assets or collect vulnerabilities related to these assets from vendors and Computer Security Incident Response Team (CSIRT). However, due to the fact that each vulnerability is related to one asset and that assets are interconnected, it becomes difficult though necessary for IT administrators to have a holistic picture of the cyber risks related to an information system that captures this interconnection between assets. For example, a violation of a security property (availability, confidentiality integrity) might prompt the exploitation of several vulnerabilities inherent in different assets and this might not be revealed by an informal method of analysis of individual vulnerabilities and thus might cause huge damages especially in sensitive systems such as power plants, nuclear facilities and hospitals where the fault tolerance is very low.

In a bid to provide a solution to this limitation, this paper proposes an approach to formally verify the expected security properties of IT assets in the assessment of risks.

# 4. CONTRIBUTION TO RESEARCH

The contribution of this paper is twofold:

Firstly, it proposes an approach to formally specify the expected security properties of an asset in terms of availability, confidentiality and integrity and to model the architecture of an information system incorporating the logical and physical interconnection between assets.

Secondly, formal software verification techniques are leveraged to verify the expected security properties of assets given the physical and logical topology of an information system. The proposed approach also allows the identification of attack scenario involving the exploitation of several vulnerabilities to compromise expected security properties of assets.

# 5. THE SOLUTION

## 5.1 Overview of the Framework

In an effort to provide IT managers with a framework that allows for the formal specification and verification of security properties of information systems, two keys aspects are covered by the proposed framework namely:

- A proposed methodology to formally specify the architecture of an information system ;
- A proposed methodology to formally specify the expected security properties of assets.

A framework for the specification of an information system incorporating the interconnection between assets is latter proposed.

## 5.2 Vulnerability Descriptor

An information system consists of a set of assets interconnected through physical and logical links. Therefore, the proposed framework provides a template for the specification of assets, logical link, physical link and forwarding or filtering rules. Unlike CVSS which associates the environmental metrics with vulnerabilities, the proposed framework however associates the environmental metrics with assets since these metrics are specific to the architecture of an information system which captures the environment.

### *5.2.1 Asset Model*

Asset refers to any valuable entity in an information system. Thus there are many types of assets such as computers, servers, routers, switches. Based on [2], parameters used to describe an asset include:

- Category: it refers to the type of the asset. This field can take values such as router, switch, PC, server, software, etc.
- CPE id
- Id: it is a unique identifier of the asset within an information system
- Physical interfaces: it refers to ports of a hardware asset
- Logical interfaces: it refers to virtual connection between the asset in question and other assets.
- The function *Send_packet* (packet, physical interface) that mimics the operations carried out within the asset in question when sending a packet. This function takes as parameters the packet and the physical interface through which the packet is sent ;
- The function *Receive_packet* (packet, physical interface) which mimics the operations carried out when a packet is received. It is within this function that the forwarding and blocking rules are implemented. This function takes as parameters the packet and the physical interface through which the packet is sent

In the proposed framework, the following template for asset based on [2] is suggested:

*Asset {*

*Id ;*

*Category;*

*Description ;*

*CPE_id ;*

*Send_packet ();*

*Receive_packet (); }*

### 5.2.2 Physical Link Model

In the proposed framework, a physical link refers to any physical port of a device. Therefore, the following parameters are used to describe the physical port:

- type: it refers to the type of the asset. This field can take values such as Ethernet, usb, hdmi, etc ;
- id: it represents the unique identifier of a port in a given equipment ;
- description: describes the interface ;
- status: it represents the status of the interface and so it can take values such as enable, disable ;
- next hop: it represents the physical port to which this interface is connected to. The value of this field will be of the type Asset.port type.id ;
- logical interfaces list: This parameter represents the list of all the logical interfaces that are bound to the physical interface in question.

The physical link can then be modeled as follows:

*Typedef physical_int {*

*Type ;*

*Id ;*

*Description ;*

*Status ;*

*Next_hop ;*

*Logical_int [];}*

### 5.2.3 Logical Link Model

In the proposed framework, a logical link refers to any virtual connection between two assets. Therefore, the following parameters are used to describe the physical link:

- id: it represents the unique identifier of the logical interface in a given equipment ;
- Source network address: it refers to the network address of the source of the link. It can be the IP address for Ethernet network ;
- Source port number: it refers to the port number used by the source for a given logical link. It can be the TCP/UDP port number for TCP/IP session ;
- Destination address: it refers to the network address of the receiver of the link. It can be the IP address for Ethernet network ;
- Destination port number: it refers to the port number used by the receiver for a given logical link. It can be the TCP/UDP port number for TCP/IP session ;
- description: describes the interface ;
- status: it represents the status of the interface and so it can take values such as enable, disable ;
- physical interface id: It refers to the id of the physical interface to which this logical interface is bound.

### 5.2.4 Packet

In a real information system, assets usually communicate by exchanging packets. Thus, in order to model the interaction between assets this paper proposes a model of packets that will be exchanged in the simulated information system. The paper considers the following parameters in the description of a packet:

- source network address: it refers to the network address of the source of the link. It can be the IP address for Ethernet network ;
- source port number: it refers to the port number used by the source for a given logical link. It can be the TCP/UDP port number for TCP/IP session ;
- source physical address: it refers to the address of the exit physical interface of the last asset through which the packet transited ;
- destination network address: it refers to the network address of the receiver of the link. It can be the IP address for Ethernet network ;
- destination port number: it refers to the port number used by the receiver for a given logical link. It can be the TCP/UDP port number for TCP/IP session ;
- destination physical address: it refers to the address of the input physical interface of the next asset to which the packet is sent ;
- Content: it refers to the content of the packet.

Packets can thus be modeled as follows:

*Typedef physical_int {*

*Source_network_address ;*

*Source_port ;*

*Source_physical_address ;*

*Destination_network_address ;*

*Destination_port ;*

*Destination_physical_address ;*

*}*

### 5.2.5 Forwarding Rule Model

Forwarding rule refers to the process that networking equipment or security appliances follow to handle a packet they receive. They are at the heart of interactions between assets. Parameters used to model a forwarding rule include:

- received-port: it refers to the id of the physical port through which a packet is received by an equipment ;
- source physical address: It refers to the source physical address of the packet ;
- source network address: it represents the source network address of the packet ;
- destination address: it represents the destination network address of the packet ;

- destination port: it represents the id of the port to which the packet is forwarded
- content: it refers to the content of the packet ;
- action: it refers to the action to be carried out when a packet with the specification that matches the aforementioned values is received. This action can be drop, forward to another asset or process by the equipment that receives it.

## 5.3 Security properties

Given that the paper is aimed at proposing a framework for formal verification of security properties, this section proposes an approach for specifying these security properties.

Usually, in model checking, there are three main types of properties:

- Safety property: It refers to properties that express the fact that undesired behavior never happen ;
- Liveness property: It refers to properties that express the fact that a desired behavior could happen ;
- Fairness properties: It refers to properties that express the fact that a particular choice is taken sufficiently often provided that it is sufficiently often possible.

In cybersecurity, there are three main security properties that must be verified to ensure the security of assets namely confidentiality, integrity and availability which will be described in the following sections.

### *5.3.1 Confidentiality*

#### *5.3.1.1 Definition*

Confidentiality is the property that expresses the fact that an information should be accessible only to authorized users or no unauthorized user should be granted access to a resource. In the model checking context, confidentiality is best expressed as a safety property. Thus, the undesired behavior that should never happen could be "an unauthorized user can access an asset ".

#### *5.3.1.2 Formal Specification*

For the purposes of this specification the following assumptions are made:

- ***send (X,Y)*** is a predicate that holds true if a packet sent by X to Y is well received by Y.
- ***is_authorized (X, Y)*** is a predicate that holds true if according to the policy, X is authorized to access Y resources.

Therefore, the confidentiality property can be modeled as:

$$G\ (\neg\ is\_authorized\ (x,y) \rightarrow \neg\ send(x,y))$$

### *5.3.2 Integrity*

#### *5.3.2.1 Definition*

Integrity is the property that states that an asset should never be modified by any unauthorized user or that no unauthorized user should ever modify a resource. In the context of model checking this property is best expressed as a safety property. In this light, the undesired behavior that should never happen could be "During the transmission of a message between two users, an unauthorized user or a malicious actor can never modify the message"

#### *5.3.2.2 Formal Specification*

For the purposes of this specification the following assumptions are made:

- ***Modif (X,Y)*** is a predicate that holds true if the user X modifies the resource Y**.**
- ***is_authorized_modif (X, Y)*** is a predicate that holds true if according to the policy, X is authorized to modify the resource Y**.**

Therefore, the confidentiality property can be modeled as:

$$G(\neg\ is\_authorized\_modify\ (x,y) \rightarrow \neg\ Modify(x,y))$$

### *5.3.3 Availability*

#### *5.3.3.1 Definition*

Availability is the property that expresses the fact that an asset should be accessible when needed to all authorized resources. In the context of model checking, availability can be expressed both in terms of safety property of the sort "All authorized users should never be denied access to a particular resource" and in terms of liveness property of the sort "Each authorized user may get access to an asset sometime".

#### *5.3.3.2 Formal Specification*

For the purposes of this specification the following assumptions are made:

- ***send (X,Y)*** is a predicate that holds true if a packet sent by X to Y is well received by Y.
- ***is_authorized (X, Y)*** is a predicate that holds true if according to the policy, X is authorized to access Y ressources.

Therefore, the availability property can be modeled as:

$$G\ (is\_authorized\ (x,y) \rightarrow send(x,y))$$

# 6. CASE STUDY

## 6.1 Description

For illustrative purposes, some security properties of an information system whose architecture is depicted in the figure below will be verified.

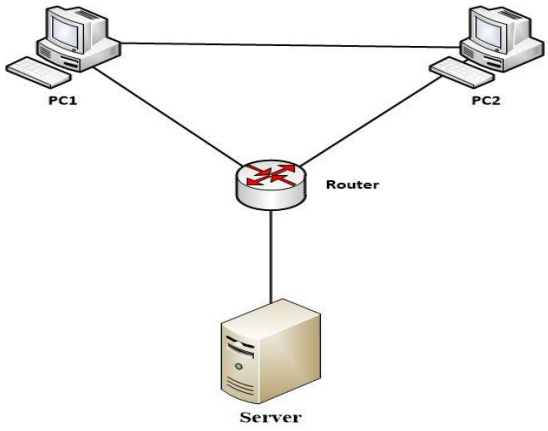


**Figure 1: Case study architecture**

### *6.1.1 Assumptions*

It is assumed that the security policy enforced in the said information system contains the following rules:

- PC1 should never get access to the server
- PC2 could access the server
- PC1 and PC2 could communicate with each other
- PC1 could gain access to the management interface of the router
- PC2 should never gain access to the management interface of the server
- PC1 and PC2 could communicate with the router

It is also assumed that the router is vulnerable to an exploit EXP1 which when sent through the management interface can cause a crash or a denial of service.

### *6.1.2 Formal Verification of Security Properties*

The interactions of the different components of the information system depicted above with respect to the rules implemented can be illustrated using the following automata.

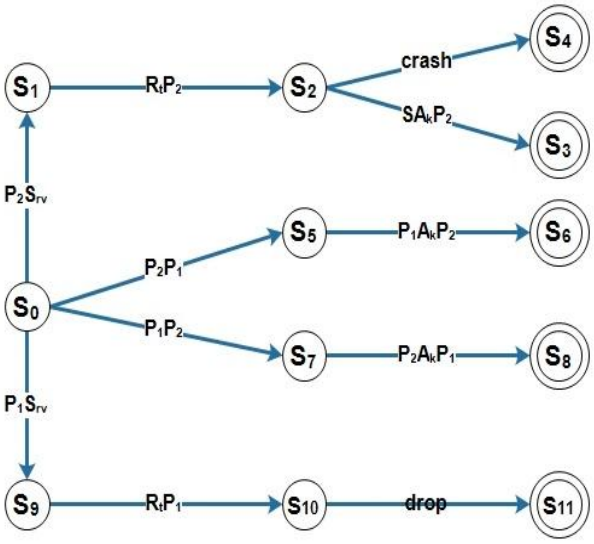


**Figure 2: automata representation**

The different states of this automata are as follows:

- S0: Initial state
- S1: PC2 has sent a packet containing the destination IP address of the server to the network. *P2Srv* holds true in this state
- S2: The router receives a packet containing the destination IP address of the server from PC2. *RtP2* holds true in this state
- S3: The server receives a packet originating from PC2. SAkP2 holds true in this state.
- S4: The router crashes after receiving an exploit code. *crash* holds true in this state.
- S5: PC2 sends a packet to PC1. P2P1 holds true in this state.
- S6: PC1 received a packet directly from PC1. *P1AkP2* holds true in this state.
- S7: PC1 sends a packet to PC2. P1P2 holds true in this state
- S8: PC2 received a packet directly from PC1. *P2AkP1* holds true in this state.
- S9: PC1 sends a packet containing the destination IP of the server to the network. *P1Srv* holds true in this state
- S10: The router receives a packet containing the destination IP of the server from PC1. *RtP1* holds true in this state
- S11: The router drops the packet. *Drop* holds true in this state.

The corresponding automata is then ATM (Q, E, T,Q0,λ) where:

*Q={S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11}*

*E={p1srv,p2srv,rtp2,sakp2,p2p1,p1akp2,p1p2,p2akp1,rtp1,drop, crash}*

*T={(S0,p2srv,S1), (S1,rtp2,S2), (S2,sakp2,S3), (S2,crash,S4), (S0,p2p1,S5), (S5,p1akp2,S6), (S0,p1p2,S7), (S7,p2akp1,S8), (S0,p1srv,S9), (S9,rtp1,S10), (S10,drop,S11)}*

*Q0=S0*

*λ= S0= {S0}, S1={S1}, S2={S2}, S3={S3}, S4={S4}, S5={S5}, S6={S6}, S7={S7}, S8={S8}, S9={S9}, S10={S10}, S11={S11}*

The approach presented in previous sections is now used to verify the following security properties:

P1: There is no way that PC1 could get access to the server

P2: The Router will be always available

P3: PC2 can always get access to the router

## 6.2 Verification

### *6.2.1 P1: There is no way that PC1 could get access to the server*

P1 is a confidentiality property and therefore can be expressed as: ***G (p1srv → drop)***

After running the verification of this property in SPIN, an error was obtained which means that the property cannot be verified as depicted in the following screenshot.

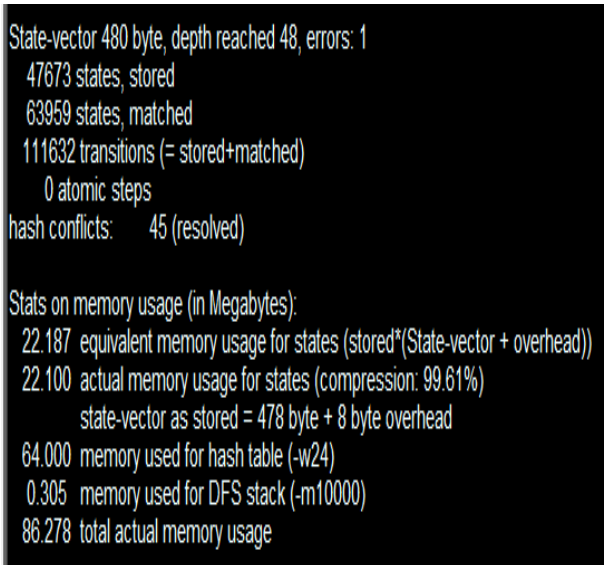


**Figure 3: verification of property P1**

In fact, PC1 could pass through PC2 to send a packet to the server.

### *6.2.2 P2: The Router will be always available*

P2 is an availability property and therefore can be expressed as: **G ¬ *crash***.

After running the verification of this property in SPIN, an error was obtained which means that the property cannot be verified as depicted in the following screenshot.

```
State-vector 480 byte, depth reached 48, errors: 1
   97463 states, stored
   68482 states, matched
  165945 transitions (= stored+matched)
       0 atomic steps
hash conflicts:       80 (resolved)

Stats on memory usage (in Megabytes):
  45.730  equivalent memory usage for states (stored*(State-vector + overhead))
  45.421  actual memory usage for states (compression: 99.32%)
          state-vector as stored = 477 byte + 12 byte overhead
  64.000  memory used for hash table (-w24)
   0.343  memory used for DFS stack (-m10000)
 109.558  total actual memory usage
```

**Figure 4: verification of property P2**

In fact since the router can receive a packet containing the exploit code EXP1, it can crash.

### *6.2.3 P3: PC2 can always get access to the router*

P3 is also an availability property and it can be expressed as: ***G(p2srv → sakp2)***.

After running the verification of this property in SPIN, no error was obtained which means that this property is verified as depicted in the following screenshot.

```
State-vector 484 byte, depth reached 213, errors: 0
  2055870 states, stored
  3397881 states, matched
  5453751 transitions (= stored+matched)
       0 atomic steps
hash conflicts:    101333 (resolved)

Stats on memory usage (in Megabytes):
  980.315 equivalent memory usage for states (stored*(State-vector + overhead))
  963.057 actual memory usage for states (compression: 98.24%)
          state-vector as stored = 475 byte + 16 byte overhead
   64.000 memory used for hash table (-w24)
    0.343 memory used for DFS stack (-m10000)
    3.487 memory lost to fragmentation
 1023.914 total actual memory usage
```

**Figure 5: verification of property P3**

## 7. CONCLUSION AND FUTURE WORK

Due to ubiquity and the widespread use of Internet and ICT, cybersecurity has become a major concern for governments as well as for private companies. Governments and critical infrastructures (power plants, water plants, etc.) depend heavily on ICT.

Therefore, given the high level of complexity of information systems in general and critical infrastructures in particular, there is a need for a system that can proactively identify in a formal and automated manner the attack scenarios that can target these infrastructures.

In a bid to secure IT infrastructure, IT security managers generally start off by developing clear and concise security objectives. These objectives are latter implemented in most cases by manually deploying and configuring security equipment such as firewall, IDS and IPS, which help detect or prevent threats.

The fact that information systems are made up of complex logical and physical interconnections between assets, coupled with the fact that vulnerability scanners and repositories generally address vulnerability from a standalone asset standpoint, ignoring the interconnected nature of information system, renders the manual approach described above inappropriate in the context of information security, hence the need for formal methods.

Model checking which has been used extensively to verify key properties of critical and complex software and electronic systems in an effort to prevent undesired behaviors that can cause severe damages comes in handy in the information system ecosystem where system failure is unacceptable.

Therefore, the approach proposed in this paper is twofold, firstly, it proposes a way to formally specify the logical and physical interconnections and interactions of IT assets of an information system and secondly, it proposes a formalization of security properties and a process to formally verify the expected security properties of the information system in question using model checking so as to ensure that assets configuration effectively allows the enforcement of expected security properties or security objectives.

The behavior of IT assets was captured in some functions *send_packet* and *receive_packe*t that mimic the operations carried out within these equipments when sending or receiving packets namely the forwarding and blocking rules of networking and security devices that play an important role in the security of an information system.

The methodology was then illustrated using a case study of an information system where some properties were checked using the SPIN tool. During the case study, it appears that modeling the components of an information system and their behaviors using SPIN was a bit cumbersome.

Future work could therefore include the realization of a benchmark of existing model checking tools so as to identify the best that fits the methodology proposed in this paper and then improve it in an effort to obtain a dedicated tool for information system security.

## 8. REFERENCES

[1] Patrice Godefroid, Koushik Sen. 2018. Combining model checking and testing in Handbook of model checking pp 613-649

[2] Ebot Enaw, Djoursoubo Pagou. 2018. A Conceptual Framework for the Design of a Nationwide Cyber-Risk Monitoring System in International Journal of Computer Applications volume 181 number 17.

[3] Adam Michael Houser. 2018. Mental Models for Cybersecurity: A Formal Methods Approach in A dissertation submitted to the Faculty of the Graduate School of the University at Buffalo, State University of New York

[4] Dirk Beyer, Sumit Gulwani, David A. Schmidt, 2018. Combining Model Checking and Data-Flow Analysis in Handbook of Model Checking

[5] Alessandro Cimatti, Ivan Stojic, Stefano Tonetta. 2018. Formal Specification and Verification of Dynamic Parametrized Architectures in International Symposium on Formal Methods pp 625-644

[6] Hamed Nemati, Christoph Baumann, Roberto Guanciale, Mads Dam. 2018. Formal Verification of Integrity-Preserving Countermeasures Against Cache Storage Side-Channels in International Conference on Principles of Security and Trust pp 109-133

[7] Jeffrey Voas, Kim Schaffer. 2016. Insights on Formal Methods in Cybersecurity in IEE computer society, issue N°5 pp 102-105

[8] Md Tawhid, Bin Waez∗, Juergen Dingel, Karen Rudie. 2013. A survey of timed automata for the development of real-time systems in Computer science review