MODELING VULNERABILITIES IN CYBER-PHYSICAL SPACES

by

KEITH MCVEY

B.S., Kansas State University, 2012

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Arts and Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
Eugene Vasserman

Abstract

There is continuing growth in the need to secure critical infrastructures from malicious adversaries. These adversaries can attack systems from different forms. They can physically break in and steal something important, or they can attack from the cyber realm in order to steal critical information. This project combines the modeling process for physical spaces along with a logic reasoning tool that can identify the state of a networked device in order to analyze large enterprise systems for combined cyber-physical vulnerabilities. Using a pure model checker would not be able to handle the near infinite states that a computer or networked device may be in. Therefore this new approach combines the use of a logic analyzer tool that with a well-defined set of rules that reasons about the security and trustworthiness of devices in the model. While there has been long study of how to secure a building from intrusion, and much research about defense against cyber attacks, there is always a large gap between the two in practice. This approach may no longer be sufficient against today's adversaries and offers little to no defense against insider threats. Combining the two in this new form allows for a more complete security view and protection against more advanced adversaries. Then this thesis shows how this approach meets a series of requirements for an effective vulnerability analysis. This is achieved by executing a model based on a real world facility with a series of induced faults that would on their own not be enough to be a vulnerability but tied together would have series consequences. This thesis shows how this approach can then be used to detail potentially unseen vulnerabilities and develop fixes for them to help create a more secure facility.

Table of Contents

List of Figures

Dedication

To all my friends and family, for the years of support.

To all my professors and advisers, without you I was dead in the water.

Chapter 1 - Introduction

Many of today's businesses are becoming more and more reliant on technology. It speeds up progress and increases productivity. It can make things easier for storing records and information. As the world becomes digital the need to protect that digital information grows tremendously. While some companies have developed strategies to guard against online theft, that is not always sufficient if the data is worth enough to attackers. If driven enough physical break-ins are still a real possibility. Now, with the integration of electronic safeguards and defense mechanisms in the building, attackers have new avenues to explore when attempting to steal. This is why a new approach is needed to look at both the physical realm in conjunction with all the electronic components scattered throughout the building for possible attack paths that could be exploited. This thesis presents a new method and tool that would scan a facility and its housed network components for just such possible attacks.

1.1 Motivation

There is continuing growth in the need to secure critical infrastructures from malicious adversaries. These adversaries can attack systems in different forms. They can physically break in and steal something important, or they can attack from the cyber realm in order to steal critical information. Physical security has always been a bit of a priority. Security is one of the reasons companies store their systems in a building often behind locked doors. Keeping a malicious adversary from actually getting to the system is an easy concept to understand and relatively easy against unsophisticated adversaries. However as the opponents grew smarter and more adept at intrusion the need to adapt physical security became more important. The Department of Homeland Security [NSP03] called for an assessment of the physical security of the nations critical infrastructures in 2003. However updating physical security is no longer enough. Much of today's world is digital meaning more and more of our lives are added to computers. This gives adversaries countless new channels to attack. A cyber attack can be less risky for an adversary since it can be harder to prove the identity of the adversary. It can also be done remotely from across the globe. Many critical infrastructures have some forms of cyber defense. However it is often not enough. The adversaries have grown more sophisticated and defenses have not grown to match. The same year, 2003 [NSS03], the Department of Homeland Security also released and initiative to assess the nations cyberspace. Both of these documents from the Department of Homeland Security show that there is a great need for both physical and cyber security assessments. However, they are stand-alone items and do not correlate one to the other. While both are important alone, combined they are true security. Adversaries will look for a weakness in one to circumvent a strength in the other. Critical infrastructures cannot afford to lack in either regard. We need the ability to analyze systems and their housing for vulnerabilities.

1.2 Research area

This Thesis is about using automated reasoning techniques to discover vulnerabilities in cyber-physical spaces. Specifically, it explores the combination of *model checking* and *network vulnerability analysis* to detect combined cyber attacks and physical vulnerabilities in large enterprise level network systems and the building they are housed in.

Model checking

Introduced in the early 1980's [Cla08], model checking is a fast, no-proof, temporal logic tool that has been widely used to great success for verifying finite state systems. However, the major problem with model checking is the *state explosion problem*, which states that the complexity of the model grows exponentially by the number of amount of components in the model. This becomes less of a problem with the advance of hardware, but generally limits the number of states in a model yet is the driving force behind most model checking research. The next big advance in model checking was *symbolic model checking*. Using binary decision diagrams symbolic model checking explores the search space rather than through individual states [McM92]. This technique allows for a much larger state size; however, this is still insufficient for large-scale enterprise systems. To allow for a much larger amount of states a new model checking technique called *bounded model checking* [BCCZ99]was introduced in the 1990's. This new form of model checking uses Boolean decision procedures to find *counter-examples* extremely quickly. Counter-examples are created to show an execution trace that shows when the specification of a model is broken.

Network vulnerability analysis

The Kuang System [Bal88] was one of the first automatic vulnerability analysis systems. Introduced in the late 1980's, Kuang is a formalization of the UNIX system security semantics. This system defines the semantics as a series of rules, then searches through them to find vulnerabilities in the system. The COPS [FS91] security checker was a tool that implements the Kuang system in order to scan for vulnerabilities such as file permissions, FTP configurations, and password strength. However, this approach does not work for more than one system. NetKuang [ZL96] is an extension on the original Kuang system in order to address a network of UNIX computers. These methods are limited against today's security standards and are considered outdated. A more flexible method was needed. In 2000 a model [TL00] was proposed that describes the abstract concept of an attack. This technique known as *Requires/Provides Modeling* takes a particular series of capabilities in the scenario for the show the attack concept is possible. This form of modeling is capable of describing more than just single attack-signatures and allow detailing of much more complex attacks. This technique started the attempt at using model-checking for network vulnerability assessment. However, a single machine creates an extraordinarily large amount of states, and with every additional machine in a network causes the *state explosion problem* of model-checking to occur.

1.3 Objectives

In this Thesis we discuss an approach to modeling network systems along with the physical layout of its housing. This method allows critical infrastructures to examine their systems for key cyber attacks along with showing faults in the physical security that could lead to potential threats. The assessment model is designed to show where adversaries can gain advantages. While this tool focuses on single locations it may be expanded to handle multiple locations with relative ease. This approach uses a model checking tool to analyze and verify routes through the physical layout and defenses of a building in order to achieve a specific goal. It checks for paths from defined starting locations that adversaries can travel. For the adversary to achieve its overall goal breaking physical security may not be enough or the adversary could encounter a physical defense that they cannot overcome. In this case the adversary may be able to employ a cyber vulnerability to circumvent the physical security. This is where the model checker uses a logic based reasoning tool in order to reason about cyber attacks. Such as, combining a cyber vulnerability to change entries in a database so that the adversary may use a forged key card to gain entry into a restricted area where they then steal a hard drive with critical information. This is a simple yet effective example of a cyber physical attack. This assessment can then be used as a base for future improvements to both cyber and physical security.

Often security is approached with the idea that attackers are external adversaries coming from the outside. However, that is no longer always the case. Rogue employees and insider threats are now one of the most common problems with security. When an insider decides to turn they are at a distinct advantage compared to an external foe. The credentials the insider has can allow them to bypass a number of the defenses that would normally halt intrusion. They may have access to restricted areas or information. This method has a unique advantage in that it is easily tailorable to model these threats as well. Adjusting the starting point of the adversary and the rules for the cyber aspects they have access too allows this tool to model insider threats. As adversaries grow more sophisticated defenses against them must also grow. This tool gives a new look into combined security against well developed foes.

Chapter 2 - Requirements

2.1 Model checker

Model checking has been used to systematically verify temporal properties about the states of a system. There have been many different model checking software developed for different applications and have been used in vulnerability assessment before. For our uses the model checking software needs a number of properties to distinguish it from the vast number of readily available model checkers. Our model checker needs to have the ability to generate counter-examples, have the ability to order statements, and should be easy to use which includes a graphic user interface and a descriptive input language.

Ordering statements

As the model checker advances through the model, the state of the model before then next temporal event is analyzed and verified for correctness. This state could break an assertion for that state of the model but not the entire model. The model checker needs to verify the current state before and after each state transition. In our model, the adversary is described as an actor roll. The actor is given a list of abilities that they can preform to attempt to break the security of the system. Having a model checker that uses ordering statements allows the model checker to dynamically attempt paths that the actor might preform at each state. This also allows the actor to use those different abilities interchangeably against each stage of the model. If the model checker did not have this feature the model would have to be changed every time the actor wished to use a new ability. Without this feature, the actor would only be able to use that first ability listed against every defense. Since this tool is intended to give viable results to the user, the tool needs to be able to return results within a reasonable amount of time. Therefore this feature is essential for our model checker to be fast enough to return results especially as the size of the model increases.

Ease of use

The tool is intended to help large business facilities. In today's world, many businesses use large networks but may not have many employees that are skilled in the areas needed to run and understand a complex tool. Therefore the tool needs to be able to be run in its entirety by people who may not fully comprehend model checking or vulnerability analysis, as well as display its results in a manner that is useful to someone who is not a security expert. For this reason, a model checker with a graphical user interface was needed. Having an easy to use interface in which a user can manipulate the tool in helps unskilled users obtain meaningful results from the tool.

For facilities of this nature and size, security is not a single person operation. Often when a vulnerability is discovered it has to be reported up to an employee's superiors and a fix must be discussed the vulnerability. Therefore this tool should be able to display results in such a manner that it is easy to see what the vulnerability is and the parts of the system that are affected. The tool needs to have a display for the results of the model analysis that shows more than just textual feedback. It should be descriptive in what nodes in the model are considered to be vulnerable. For these reasons, the tool should have a graphical output display as well.

Counter-example generation

As the model checker iterates through all the states in the model, it validates the model against any specified condition that the model is still correct. In this project the tool should be able to create a path through the model if one exists that shows a vulnerability in the system. For the model checker this means the software should be able to generate a path if there is a condition that violates the model as a secure system. Since this tool is designed to not only find faults, it also needs to show the user how the model is vulnerable. This feature is easily implemented as a counter-example. In our scenario the model shows the physical layout of the building that houses the network system along with the cyber components. With the properties of the building laid out in the model the model checker will determine if there are vulnerabilities that allow an adversary into an area that they should not have access. Without the counter-example generation the model would simply break and not give any viable results to the user. The counter-example will detail to the user exactly what steps the adversary needs to take within the model to circumvent the defenses. This requirement works for any starting point inside the model and any adversary model as long as the user specifies these conditions when the model is created just like the security assertions. Counter-example generation is the primary requirement for the tool to give usable results to the user.

Input language

Model checkers have been used for vulnerability assessment before. However, a major limiting factor in the effectiveness of model checking is the limitations of the input language. The model checker can only give results based on the model that is used as input, and this model must be written in the specific language that the model checker can understand. For our tool we wanted a model checker that used a language where the user could be as specific as possible and describe the most detail into the model as possible. The language needs to be adaptive enough to describe both the layout of the building and the physical properties of the room components.

2.2 Network vulnerability analyzer

Network analysis is a very difficult problem to attack. Large networks can have many computers and components linked together. There have been recent attempts at modeling vulnerabilities on single hosts. Even if used in parallel with multiple hosts, this will only locate vulnerabilities that are on a single machine and may not have much impact on the overall system as a whole. However, the ideas behind the interactions between components on a single host can be similarly used for network interactions. For our project we need a component to analyze the cyber components of the model. This component needs to be able to formally recognize specific vulnerabilities, and needs to be able to be configurable to the different specifications of the network.

Vulnerability modeling

When penetrating a network, most adversaries take advantage of a vulnerability that is on one of the host machines. According to the Computer Emergency Response Team (CERT) [CER14] a vulnerability is defined as " a software defect that allows an attacker to violate an explicit (or implicit) security policy to achieve some impact (or consequence)". These vulnerabilities are usually documented to particular online cyber security authorities such as CERT and the National Vulnerability Database (NVD). Since the fight for cyber defense is an always changing and never ending field, the model needs to be able to check such authorities for these reported vulnerabilities and incorporate them into the model checking. However, these are not the only authorities for vulnerabilities and the ways that vulnerabilities are described between the authorities can vary drastically. This means that either the user must use only certain vulnerability authorities and limit the number of reported threats or manually translate the vulnerability definitions. Having an analyzer that can interpret different vulnerability definitions from multiple sources would increase the tool's effectiveness by potentially covering a larger amount of vulnerabilities.

Configurability

Large networks can be very complex having multiple hosts and devices. Therefore there can be nearly unlimited number of combinations having vastly different setups. For our project the network analysis tool needs to at least be configurable for each individual network. This will allow for the tool to have tailored results for the actual system in the model instead of a generic system. With the large number of configuration possibilities the correlations between each component may not be apparent to all users immediately and missing a connection could lead to the tool finding a vulnerability in the system that does not exist or even worse falsely declaring the system secure when there is a vulnerability actually exists. Having a analyzer that can automate at least part of the configuration process would decrease the amount of work that the user needs to do and would increase the reliability of the tool.

2.3 Adversary model

The battle for security is a race to close vulnerabilities faster than the adversary can find them. For our tool the adversary can vary from model to model. However all adversaries are measured by the model in the same way. They are defined by their capabilities. It does not matter to the tool why an adversary is trying to break into the system or facility; it only needs to know what the person can or cannot do. This means detailing all of the physical capabilities such as the ability to pick a lock or climb a fence to the model against the physical properties of the components inside the facility. This also means listing the tools that the adversary has at their disposal such as if they have a way of cutting through padlocks or breaking door hinges. The description of an adversary's physical abilities is not sufficient in this tool since it also considers the network components spread across the facility. Therefore the adversary must also be described by their privileges in the network. While initially their privileges may be minimal or none this can change throughout the attack. The adversary model must be able to adapt as the adversary progresses.

Chapter 3 - Design

This project, named Icarus, is made up of 2 distinct parts. The first is a model checking software that analyzes the physical properties of the building the network is contained in. The second component is a logic based network vulnerability analyzer. This component in the Icarus project handles any network devices and the cyber components inside the model. The two components combine to form the Icarus tool. The physical components of the model are checked by the Bogor model checker. Then any network components are treated as a black box inside the model and is passed to the network vulnerability analyzer MulVAL where the state of the network is determined and passed back to the model checker.

One might wonder why it was decided to separate these functionalities between two different parts instead of using one unified structure. First of all, a model checker brute forces all possible states in a model. For such a tool to verify all the potential states for a single computer with just one user would take a substantially long time. Then if it were used to verify large complex networks, even the most advanced model checking software would near orders of the lifetime of the universe to complete [EME08]. Therefore a model checking software could not be used alone. Secondly, the underlying structure of MulVAL could possibly have been used to verify the state of a physical system. However, this would require a great restructure of the program itself and the creation of very specific rules to complete. This strategy was simply beyond the scope of this project when there were many readily available model checkers that could be used with much more ease to complete the same task.

3.1 Model checking software

The majority of Icarus' capabilities are done by the model checker to verify the state of the network's housing. The building is described in the models by physical properties of each room and the items within. Each room has a path of egress which is described by the methods of entry. The inside of the room is modeled by anything encased inside the room along with any wiring or ducting that runs through that room's walls or ceiling. In this manner the model shows anything the adversary can gain access to once inside the room. A number of different model-checking software were considered for the Icarus project: Alloy, SATMC, and Bogor. Each of the model checkers has unique advantages that could be useful for the project.

Model Checker	counter-example Generation	Ordering Statements	Input Language	GUI	Used previous for security
Alloy	Yes	No	Alloy	Yes	No
SATMC	Yes	Yes	SATE	No	Yes
Bogor	Yes	Yes	BIR	Yes	No
Slam	Yes	No	SLIC	No	No
BLAST	Yes	No	C	Yes	No
NsSMV	Yes	Yes	NuSMV	No	No

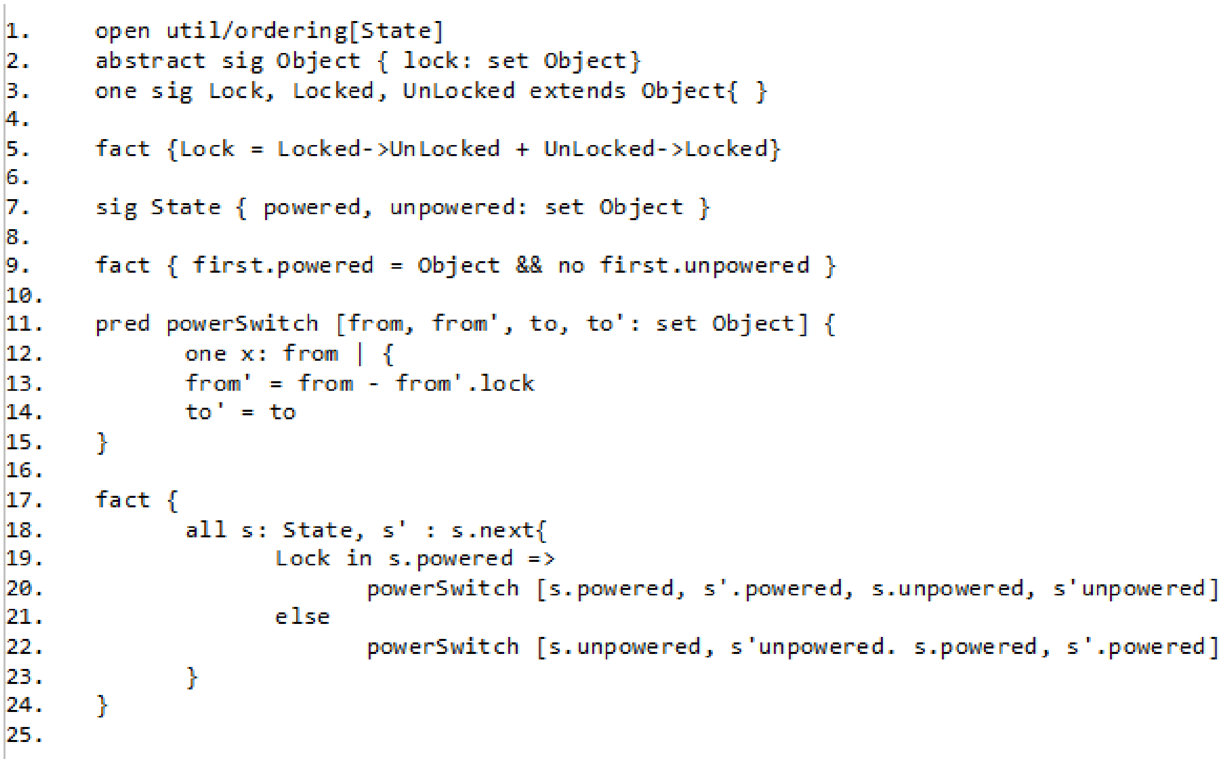
Figure 3.1 Model checkers

SATMC

SATMC [COM05] is a Boolean satisfiability based model checker. It was designed to be a flexible bounded model checking software for security protocols. Developed in Prolog the model checker was intended to be a modular piece of software that could be readily adapted to suit changes in model checking techniques. SATMC uses the SATE language to represent the system based on an Intermediate Format (IF) vulnerability specification. With this information the SATMC model checker would run through the model and output a results page similar to figure 3.4. While the information in this output is great for analysis, our project wanted a visual component that could easily show inexperienced users what an attack would take. It is worth noting that SATMC could have a GUI if coupled with the AVISPA tool. However, this was not the biggest issue SATMC would pose to the Icarus project. This tool was intended to model single cyber security protocols. It was not intended to describe the physical world. The SATE language used in SATMC could not describe the layout of a building in the model or the physical properties there of.

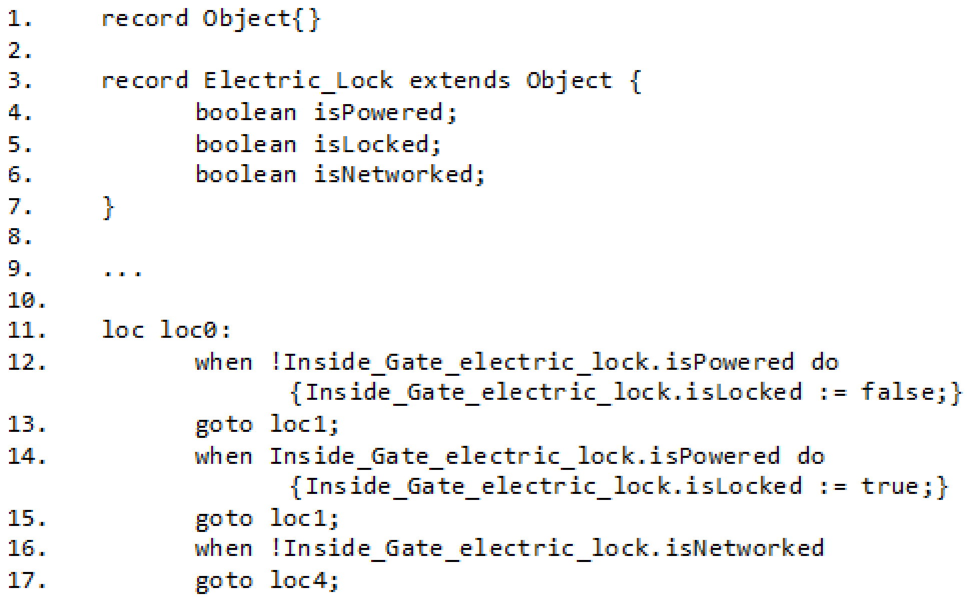
Alloy

Originally Alloy was first selected as a candidate. Alloy is based on an actor utilizing an order of actions on the objects inside the model and facts about the objects [ALL14]. Facts are unbreakable rules that the model must follow and control the way objects interact. Then the model is verified by a list of assertions. The assertions are rules about the state of the model. If all of the actions are carried out and none of the assertions are broken the model is considered valid. However, if through any combination of the actions causes the model to be in a state that breaks any assertion, this is shown as a counter-example. Alloy's language is not very intuitive and can be difficult to implement. Figure 3.1 is a code example showing an electric lock. The lock switches from the state of having electrical power to not having power. When the lock is not receiving an electric current it unlocks. One can see the sheer volume of code necessary to model such a small occurrence. Using this model checker would prove to be difficult in large models since any case that would cause the model to be vulnerable would require an assertion. Therefore the list of assertions for any reasonable model would be unfeasibly large and prone to errors.



```
open util/ordering[State]
abstract sig Object { lock: set Object}
one sig Lock, Locked, UnLocked extends Object{ }

fact {Lock = Locked->UnLocked + UnLocked->Locked}

sig State { powered, unpowered: set Object }

fact { first.powered = Object && no first.unpowered }

pred powerSwitch [from, from', to, to': set Object] {
        one x: from | {
        from' = from - from'.lock
        to' = to
}

fact {
        all s: State, s' : s.next{
                Lock in s.powered =>
                        powerSwitch [s.powered, s'.powered, s.unpowered, s'unpowered]
                else
                        powerSwitch [s.unpowered, s'unpowered. s.powered, s'.powered]
        }
}

```

Figure 3.2 Electric lock code example in the Alloy modeling language



```
record Object{}

record Electric_Lock extends Object {
        boolean isPowered;
        boolean isLocked;
        boolean isNetworked;
}

...

loc loc0:
        when !Inside_Gate_electric_lock.isPowered do
                {Inside_Gate_electric_lock.isLocked := false;}
        goto loc1;
        when Inside_Gate_electric_lock.isPowered do
                {Inside_Gate_electric_lock.isLocked := true;}
        goto loc1;
        when !Inside_Gate_electric_lock.isNetworked
        goto loc4;
```

Figure 3.3 Electric lock code example in the BIR modeling language

Bogor

The Bogor model checker was designed with four principles in mind: Direct support of modern language features, extensible modeling language, open modular architecture, and encapsulation [BOG05].

For most model checkers such as NuSMV, SLAM, and *BLAST* , the input language determines the abilities of the model checker itself. Bogor uses a version of Bandera Intermediate Representation (BIR) [BOG05], which was originally designed to translate Java programs to the specific input language for some already existing model checkers. However, in this form BIR does not support some of the features of modern software. The revised version of BIR extends the language so that it includes generic types and polymorphic functions. With the new extended version BIR now supports dynamic unbounded heap and thread object creation with automatic garbage collection. These properties give Bogor the ability to support more modern languages. The extension ability of BIR also allows users to create new types, expressions, and commands. Any extension declaration needs two things. First a signature declaration is needed to show any new symbols along with the arguments that will be used. Secondly it needs the name of the package in the project that has the implementation of the semantics for the new extension. This allows users an easy way to implement any domain-specific components into the model. With this extendable language the Bogor model checker language would allow the user to describe the physical layout of a system housing. Figure 3.3 shows a code segment of an electrical lock. It is very easy to understand and implement since it acts very similar to many popular programing languages. In comparison to the Alloy language in figure 3.2, the BIR language can be simpler and has the capability to easily extend objects' functions in relatively small amounts of code.

The Bogor architecture is designed to be modular. It is divided into three different portions: front-end parsers, the BIR's semantic transformation components, and the model checking engine. The implementation of Bogor's components reduces the components' dependency on each other, which allow for extension and encapsulation [BOG05].

The Bogor model checker also comes with a built-in GUI with a visual display for the objects in the model such as the rooms and components of the rooms in a building, and the path generated for counter-examples. This would easily be able to display information in a way a user can understand even if that user is not a security specialist. With its extendable language, graphic user interface, and counter-example generation, Bogor was chosen as the model checker for the Icarus project.

3.2 Network vulnerability analyzer

No model checker rearguards of implementation or sophistication, can check the near infinite number of possible states that a computer can be in anywhere near a reasonable amount of time. So even Bogor would never finish checking the model if even a single electronic device is in it. Within any large enterprise system there would be a vast number of connected terminals and electronic devices through out the facility. Therefore this project needs more than just a model checker. We selected a logic based network vulnerability analyzer to fill this role known as "Multi-host, Multi-stage Vulnerability Analysis Language" (MulVAL) [OGA05] because of the proximity of expertise.

MulVAL

MulVAL uses a set of predefined logic rules to reason about the security of a network. For MulVALs evaluation it needs to know: if there are any reported vulnerabilities already on the system known as Advisories, the software that and services that are configured and running on the system known as the Host Configuration, the configuration of firewalls and routers denoted as the Network Configuration, the users of the system called Principles, how the components of the network can interact known as Interaction, and what access policies are allowed called Policies [OU05]. MulVAL implements the Open Vulnerability Assessment Language (OVAL). OVAL has definitions for many reported vulnerabilities. As of January of 2005, there are over 800 reported vulnerability definitions on OVAL across Windows, Linux, and Solaris.

Introduced into MulVAL are several predicates used as exploit rules that define what is necessary for a network component to have a vulnerability. Then using horn clauses to determine that if the certain attributes for a vulnerability. MulVAL also reasons about multi-stage vulnerabilities where an attacker can further compromise the system beyond just the initial attack. The policies for vulnerabilities can be combined into larger scale attacks against the system. Therefore with the proper layout of the system, and a rule list, MulVAL can automatically analyze networks for cyber security threats both remotely and local [OU05].

MulVAL's abilities to analyze more than one host machine for complex vulnerabilities and return a state of secure or vulnerable makes it the ideal candidate to be coupled with the Bogor model checker for the Icarus project.

Chapter 4 - Small scale results

In this thesis we first developed a small scale proof of concept scenario. This scenario was designed to test the basic functionality of the tool. In this scenario we designed two series of actions for the actor to consume in order to show a valid secure model and a vulnerable model. We first used a very limited version of our adversary model in the proof of concept scenario.

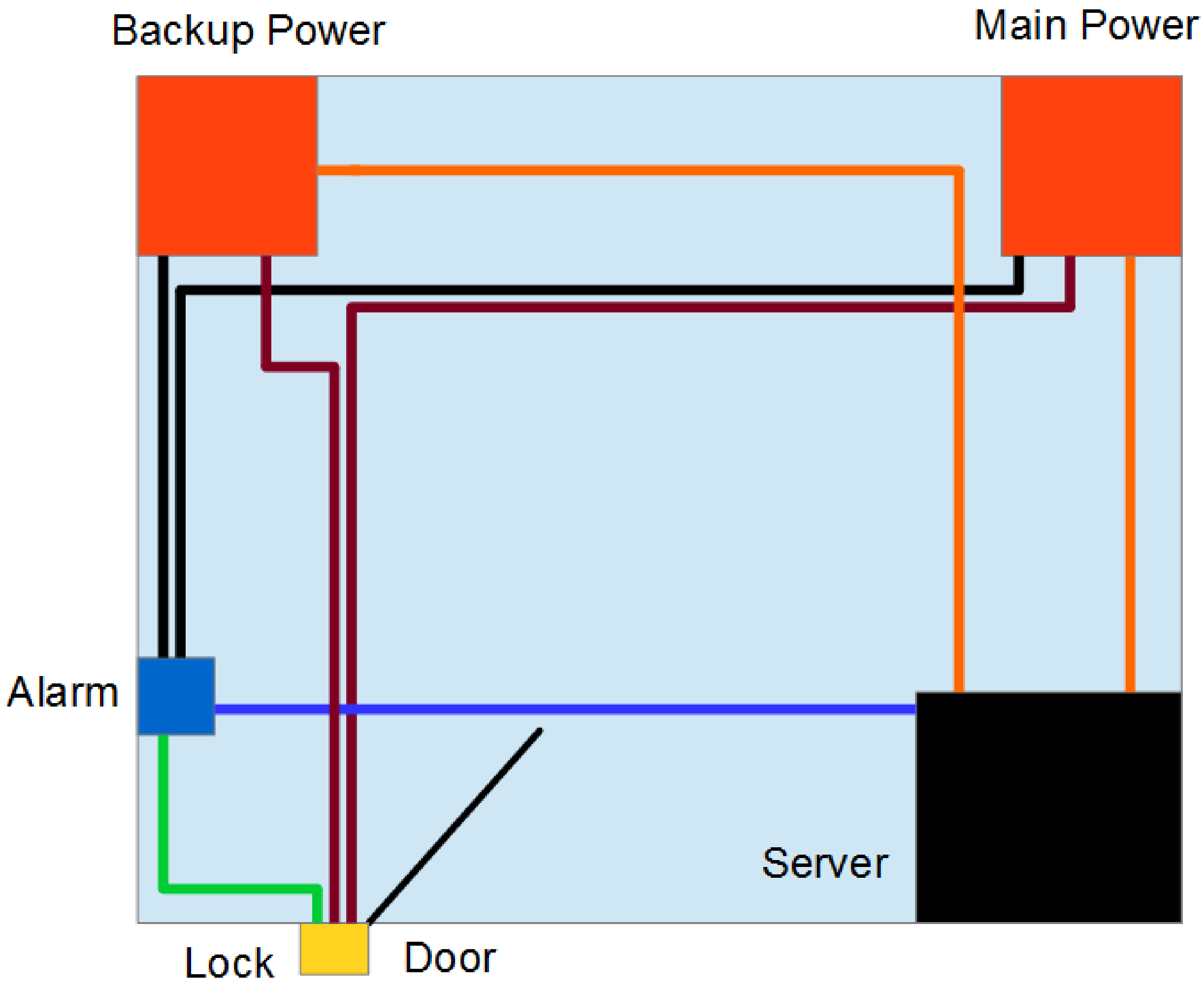


Figure 4.1 Proof of concept model for a small single room space with enclosed power and connections

4.1 The proof of concept model

This model is comprised of a single room with a single entry point. The door into the room is initially closed and locked with both an electronic lock, which must be opened by a valid key card, and the physical lock which is opened with a key. The electronic lock engages and disengages the physical lock when activated by a valid user key card. The physical lock has the property that it can be picked like any ordinary key lock. Inside the room there is a server, and an alarm. In figure 4.1 the door lock is connected to the server for authentication by the blue cable. Since there is likely to be many potential employees in the facility it would be unlikely that the door lock would hold a list of valid users. Therefore it would need to query an authority to allow for access into a restricted space. The lock is also connected to the alarm. If the door is opened using just a key and not by an authorized key badge the alarm will be tripped and alert the facility to an unauthorized access. This would not be too big of a problem for a valid employee who may have lost or forgotten their access key but still needed to gain entry to this space. However, for an adversary, alerting the entire building to your presence is not a desired effect. Figure 4.1 also shows that inside the room are two power sources: a main and a backup power supply. These power supplies fuel the electronic lock, the server, and the alarm by the red cable for main power and the yellow cable for the back up. The power sources are housed inside the room so that it is more difficult for adversaries to gain control. If the power to any of the electronic components is cut then those components will go offline. There is redundancy since this is a major concern for most facilities. The actor in this scenario has a few distinct capabilities that allow him access into this room. For the first case the user is a valid employee of the facility with a valid identification key card. The actor would approach the outside of the space, swipe his or her card at the electronic lock. The lock would then query the server to find that this user is authorized to enter this space. The electronic lock would then disengage the physical lock on the door and the user would then enter the room.

Initial implementation in Alloy

Originally Alloy was used to check the model. Alloy was given these steps as an ordering for the actor and the functionality of the components described in the model. The server was only given the ability to approve users and never reject. The assertions on this model were simple. A user cannot be inside the space without being authenticated and the alarm not triggered. This simply states that if someone gets inside the room either the alarm is going off or they are an authenticated employee.

Improvements using Bogor

This model was then reimplemented in Bogor to test the differences between the two model checkers. Bogor does not require the list of assertions for this model like Alloy. Bogor would determine its paths by the capabilities of the actor. In the Bogor model the room layout was identical however the actor was simply given the ability to swipe a key card and open an unlocked door. The Bogor model checker would iterate all permutations and combinations of the actor's capabilities in order to find paths into the room. In this model since the actor does not have any capabilities to gain access to the room other than a valid key card the model checker would verify that the space could not be accessed by the actor other than by being authenticated by the server. The model checker proved that this space is secure against this sophisticated of a user . The scenario was run again however the user was given a physical key to the room. The model checker would then show a path that the actor could take that still allowed him or her access to the space but this time triggered the alarm. However, since this is still with in the assertion that an unauthorized user cannot enter the room without the alarm being triggered the space is still considered secure.

To show a vulnerable room, the connection from the lock to the alarm was removed. In this new state the alarm will never receive the notification that an unauthorized user has opened the door. The same test cases were run on the model again. The first where the actor only has the capability of the valid key card would again pass the assertion and the model verifies as secure. However, when the user was given the physical key again and does not use his or her key card, the actor would gain unauthorized access to the room without triggering the alarm. The model checker would show this as a counter-example to the requirements of a secure space. This space is now considered vulnerable.

4.2 Introduction of cyber component with MulVAL

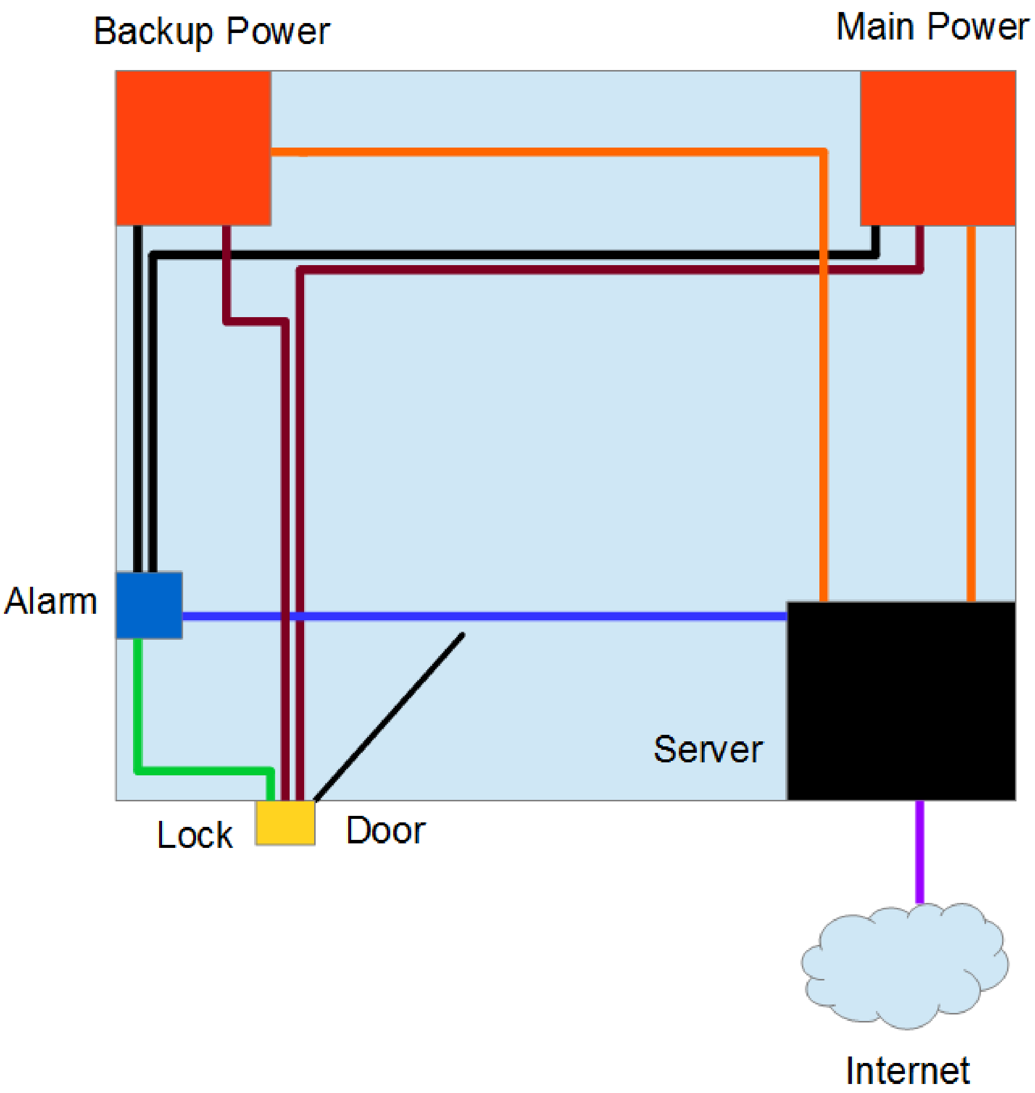
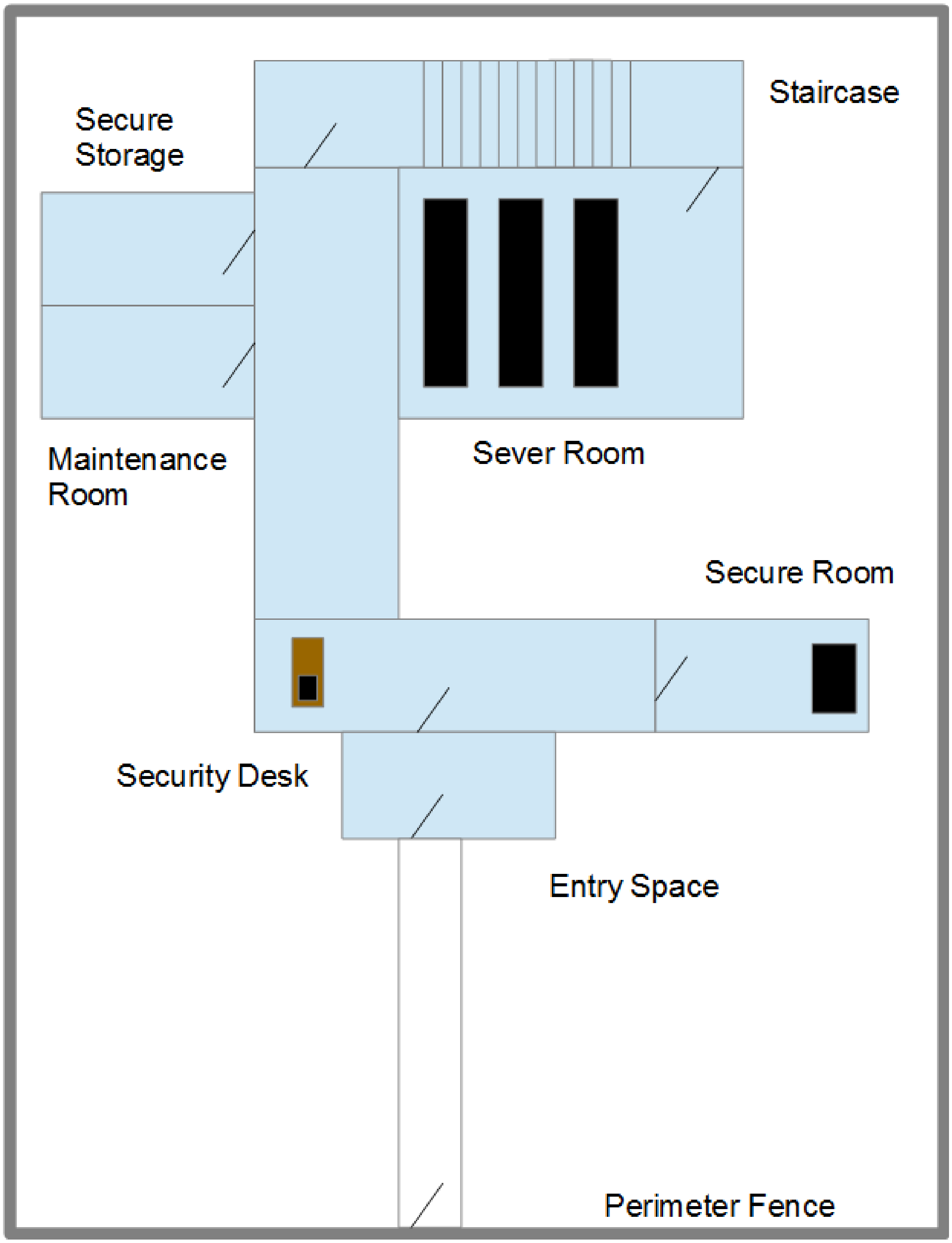


Figure 4.2 Proof of concept model of single room space wish enclosed power and connections with the addition of an external connection to the Internet

These models only show the capabilities of the model checker. The tool itself needs to show what an adversary can to a networked system that may have some vulnerabilities. In figure 4.2 we opened the server to a network. The network could be considered either the Internet or just the local network inside the facility. On this network we induced a vulnerability that allows a user to modify the credentials on the server. This would allow an attacker to change the permissions of anyone on the server's database so that they may gain access into this space when they may not have been able to before. With this setup and vulnerability in place the model checker would run up until the request for the server to authenticate. Once that point was reached, MulVAL is queried given the networks setup and advisories. MulVAL would return the vulnerability and that the system is vulnerable. Now that the model knows the server has been compromised the server would allow access to the user and they would gain entry into the space without triggering the alarm. Since the access to the room was gained with the aid of a compromised server the model itself would show the room as vulnerable.

Chapter 5 - Case study

The Icarus tool was intended to be used on a larger facility than just a single room. Therefore a new model was designed in order to test the capabilities of the tool on a larger scale. This model was designed to simulate a more real-world scenario. While there are some standards for facilities of this nature such as the American National Standards Institute's ANSI/TIA-92 [DAT05], these are guidelines for the setup of a building. These standards alone do not ensure security, they only mitigate some common risks. Even buildings that meet the same standards will have different floor plans and house equipment in different parts of the building. Therefore these buildings could have different vulnerabilities even though they conform to the same construction and security standards. For this reason I created a seven space building model (based on a real facility) that includes two floors, then ran Icarus on the new model to compare against the small scale results.



5.1 Physical seven space facility described in the model 2.0 with an included network

5.1 The model of a seven space secure facility in BIR language

Physical layout and components

The model for this scenario is a fictional creation based on exploring a real facility. It is meant to show a number of different spaces that could exist inside such a facility and ways those spaces might be exploited due to their proximity to each other. The first noticeable difference is that this space in encompassed by an external fence. The fence has only one entrance. It would be easy to imagine that many of these types of facilities to have an external barrier around the property as a first line of defense. This model expects some distance between the fence as if it was a perimeter fence; however, the tool does not differentiate between fences that are far from the building as opposed to those that directly line the building itself.

While for fire safety reasons buildings are required to have more than one point of egress, this model has only one door leading into the facility. This was chosen for simplification of the model. Directly in front of the gate in the fence is the single access point into the building. This door leads into an entry space that is completely empty except for another door leading into the main portion of the building. It is common amongst secure facilities to have such an entry space where only one of the two doors maybe open at a time. This allows access to the building while still keeping the main portion of the facility a closed system with no direct access to the outside.

Once through the second door in the entry space one would enter a main hallway for the facility. This hallway is the main thoroughfare for the entire building. Immediately to the left of the door leading into this space from the entry way would be positioned the front desk. This is where personnel entering and exiting the building would have to check in with a security guard present some form of identification. Nearly all secure facilities would have some form of logging procedure in order to keep a record of who has been entering and exiting the building at all times. Therefore it is easy to see why such a space is included in this model.

To the right of the door entering the hallway is a secure space. This space houses the video surveillance records for the building. It is also a room that during an emergency can be locked down and used as a safe space for employees. This is an important space in our model. This room holds all the records for the cameras placed around the facility. For this reason an adversary may wish to enter this space and either destroy, tamper, or take equipment or information from inside. This space would also have network cables that could potentially be used to access the network inside the facility.

The hallway in this facility extends to the right side of the desk and into the rest of the facility. Directly attached to it are 2 small spaces. These spaces could be used for a number of different purposes for the facility. It was decided to have these two auxiliary spaces in the model since inside many such facilities there will be extra rooms that have been re-purposed throughout the life time of the building. One of these spaces is set as a maintenance closest since every running building would employ a janitorial staff and need a space for their equipment. This particular implementation of this space is significant because this space would not typically have the same security standards as most others; however, its proximity to other rooms might lead to potential attacks along with providing access to tools that maybe useful to an adversary. The other of these two spaces is implemented as a storage room for secure documents. It would be easy to see that many companies would be interested in having hard copy back-ups or records that they would wish to keep. This room would be a storage room for some such type of data. This makes it a significant room in the model. Not only for its contents but also since it shares a wall with a lesser secured space. These two spaces combined would show the tools ability to reason about such conjoined spaces.

At the end of the hallway there is another door. This door leads into a stairwell to the basement floor. While this room itself has little significance, its incorporation into the model would show the tools ability to reason about multiple floors since it is reasonable to suspect that many such facilities would be multistory. Having a stairwell that leads to another floor also gives reason to pay attention to the areas between spaces such as beneath the flooring or about the ceiling panels. This is important since it maybe more difficult for an adversary to get access to cables running between spaces in the walls than in the ceiling.

The final room is connected to the base of the stair well on the basement floor. This room is denoted as the server room. It holds the computer servers running all the networked components inside the facility. This space is crucial to the model. It would need to be one of the most secure rooms and very difficult to get too. This is why it is placed the farthest inside the building in relation to the point of egress, and on a separate floor.

These seven spaces make up the model of the building. Each room is comprised of walls, a ceiling, a floor, and at least one door that leads to the next space. Inside the walls, floors, and ceiling is piping and network cables.

Network components

The Icarus Tool was designed to check in tandem both the physical layout and any networked components inside the housing. Therefore our model needs more than just the server room to properly display the capabilities intended. In this model there are a number of networked components that are connected via Ethernet cables to the server. It is important to note that in this model the network is not connected to the Internet. The decision was made that for the purpose of this demonstration, should the network have a vulnerability that leads to an attack on the system, it is less import to distinguish whether that attack was a remote vulnerability or from a local machine but that some vulnerability existed and what consequences it lead to.

The most common component throughout the entire model are electronic door locks. These locks are identical to the single lock on the small scale model. They are placed on every door throughout the facility with one exception. The door to the maintenance closet does not have an electronic look, and only contains a physical key lock. These locks need an electronic key card to unlock. An employee of the facility would present his key card to the lock, which would then verify access permissions through the server. If the user has permissions to access the space the lock would disengage and allow access. All of the locks use an independent hard line to the server so that if one lock is compromised the others remain functional.

There is one computer terminal placed in this model. It is located at the front desk. This terminal represents the work station of the security guard. It is common place for such facilities to have a computer at the front desk to allow easy communication to the security guard. In the model this shows an access point to the network that may prove to be a vulnerability. This computer has limited access rights to information on the server and therefore would not be a single point vulnerability to an adversary. However, since it has rights fitted to a security officer it might be a point of interest an adversary my take advantage of to either disengage some other security devices or attempt to gain extra access to the network.

Also in the system is an alarm. There is a single “alarm” that runs throughout the entire building, which means that like in the small scale model should a person enter a room unauthorized the alarm should sound. However, in this model the alarm does not sound for just that room, it should alert the entire facility. This also leads to the potential that if an adversary could get to the central alarm, they could disengage the entire system for every room. This was chosen to display a single point of failure in a security system.

There are two severs in this model. The main server in the basement that houses data that an adversary might covet, and the server in the secure room that holds all the video surveillance footage. It is easy to imagine any facility of this nature having video recording hardware throughout the facility. These cameras are all connected directly to this sever by independent Ethernet cables just like the door locks. However, they are not connected to the main sever. This means that the cameras do not supply access to that network. Typically someone attempting to attack this facility would not want to get caught. therefore it is easy to see why an adversary would want to get access to either this server or the cameras themselves to cover any evidence of the attack.

Since all of the networked components are electrically they need some form of power to function. Attached to each component are a main power source and a back up power source. The main power source is a hard line power connector running from the outside power grid into the facility. Typically such a facility would have more than one power source from the outside for redundancy. For this model the decision was made that should the adversary be sophisticated enough to take down a power grid, they could take down more than one and therefore showing multiple outside power sources would not add to the effectiveness of this model. Since power critically is important there is a back up power supply that is housed on site in the facility so that if the main outside power source is lost the facility would be able to keep functioning at least for a limited time. This secondary power source is a generator located within the perimeter fence but outside the building itself. For both sources power is transferred on its own independent cables to each networked component inside the facility. However, these cables run parallel to Ethernet cables to each device.

5.2 Realizing the model in Icarus

With a complete model of the facility layout and the network architecture, the model was ready for analysis by the Icarus tool. Just like in the small case scenario, the model was run in three different versions: the base model, a seeded physical vulnerability, and a seeded cyber-physical vulnerability.

For the base model, the model described in 5.1 without any seeded vulnerabilities was used, because it describes facility with high security intentions. A basic adversary with no special skills it returns as a secure facility without vulnerabilities. Then to test different functionality of the tool, vulnerabilities are seeded into the model. The same fault used in the proof of concept disconnecting the alarm from the outside door of the entry way was introduced. This is very similar to the test run on the small scale version. This version of the model is to check that simple assertions about the system hold. Secondly a network vulnerability was seeded so that the terminal could access and modify data on the main server thus testing the network analysis of the model.

State transition formalization

For the model to continue there must be some transition from state to state. Since the objects in the Icarus model are static without the adversary, the only way the state can change is if that actor imparts some change in the model such as moving from a room to another room or disabling a security device. We formalize the state transition as such:

$$\text{Loc}_k\text{: when } P_i \text{ and } A_j \text{ then } \{!P_m \dots !P_n\}$$

This translates to: at any location k when an object has property P_i and the actor has the corresponding ability A_j then properties P_m through P_n are changed in the model. Properties and abilities are defined in the model as boolean variables such that an object either has or does not have a certain properties. This can be anything such as a door is either closed or not closed. When ever a properties in the model is changed that constitutes a change in the state of the model.

Language restrictions

A major part this tool was to have an input language that was more descriptive than previous vulnerability analyzers so that it could more accurately describe the facility in question. The language in the Bogor model checker allowed for a more detailed layout of the building. Using it the model could accurately detail the physical room layout, wiring in walls and ceilings in and between rooms, and physical attributes about the doors and locks. All of this leads to a more realistic representation of the physical space that comprises the facility and leads to a better understanding of plausible real-world vulnerabilities that may exist. However, this is not perfect. The model is limited to the physical attributes that the writer specifically mentions. There maybe many of either unknown or omitted attributes that either the model writer does not know about or are not standards for the specific component of the model. While a model writer maybe able to describe the fire resistance of a door based off of the specifications given from the manufacturer, a writer cannot describe how much effort an adversary might have to apply in order to drill a hole in the door. This is problematic in that every attribute must be white listed in the model. Neither the model nor the language knows every detail of any component designated as a “door” more that the small specific attributes the writer lists. Even the most adept model writer is very likely to miss some attribute in at least one component in a sufficiently complex model. Since there is no way to know which attributes maybe relevant to all vulnerabilities in that component the model may not find a vulnerability if that attribute is omitted.

This is also the same for the adversary. While there are standards for modeling adversaries, the human mind is incredibly creative. It is not possible to describe all the actions that an adversary might be able to take during the course of a break in. Any action that the model wishes to be considered must also be white listed on the actor in the model. Even a simple actor can have a very long list of abilities even though they may not all be used for every possible attack. This leads to a very long list of abilities and it is likely that one or more has been omitted. Since it is impossible to know every potential action an adversary can take it is plain to see the limits of this language.

Counter-example generation

A main purpose of the Icarus tool is to be able to generate paths an adversary can take in order to break into the facility. This is achieved through counter-example generation. Utilizing this feature form the Bogor model checker in the seeded alarm vulnerability from the small scale analysis, the Icarus tool could find a path that would end with an unauthorized user inside a room.

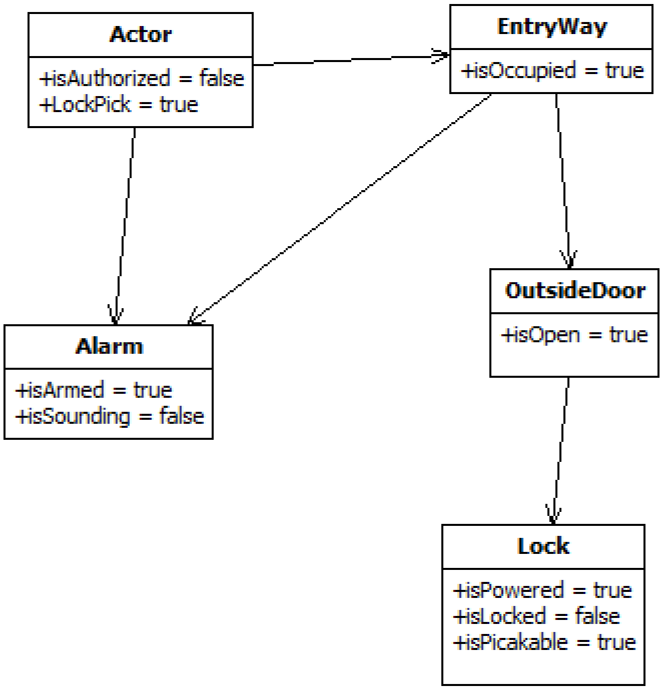


Figure 5.2 Counter-example of proof of concept vulnerability in a more complex facility

The Icarus tool was intended to display the state of the model for a particular break in. Figure 5.2 shows a reduced amount of the displayed results for a generated counter-example. This was found by the tool when there was a seeded fault into the entry way room at the front of the facility. The fault was were the alarm was disconnected from the door leading into the entry way from the outside. therefore, the adversary could pick the physical portion of the door lock, and enter the space unauthorized. As one can see the display gives a representation of some of the components an adversary has to interact with in order to gain simple entry into the space. This counter-example shows that the actor in this model was able to be inside the entry way of the facility without using the electronic lock. Since the lock has the attribute of being pick-able and the adversary has the ability to pick locks, the adversary can open the door. Then because the electronic lock is not connected to the alarm there is no notification to the facility that something is wrong. Therefore there is a vulnerability in the system.

5.3 Introducing a multi-stage cyber-physical vulnerability

Now that a baseline of Icarus has been established, it is time to test a more complex attack against the facility. A scenario was devised to see if a multi-stage cyber-physical attack would be recognized. The same building layout model as section 5.1 would be used to showcase Icarus' abilities. However, this version of the model would be in much more detail and attempt to mimic a real-world example of the facility and its network.

Adversary model

The first major difference for this test scenario is the adversary model. Before, the adversary was very limited in his capabilities and access rights. In this experiment the adversary has been upgraded with new abilities including: lock-picking, climbing, door hinge removal, knowledge of electrical wiring, and the ability to cut power and Ethernet cables without injury. The adversary also has prior knowledge of the buildings wiring schematics. Another advantage is that the adversary is an employee of the facility with basic access rights to non-critical rooms such as the barrier fence, entryway and the hallway. However the adversary does not have access to the maintenance closet, server room, storage room, or the secure room. As an employee the adversary has some access permissions on the network. These are limited to the local machine at the security desk and do not include any permissions on either server nor the security infrastructure. Lastly, a new case where the person infiltrating the facility is not the only attacker in this case. A remote accomplice has been added. They share the same basic network privileges, which means the remote adversary can log into the facilities network using the employee's credentials. This is a much more sophisticated adversary than Icarus has faced before. The malefactor has more skills, access permissions, and tools at his disposal.

Building components

The layout of the building remains the same as before. No rooms have changed location nor are there any new spaces in the building. However, more detail about each room and the components of the building is required. The details about the Ethernet and power cables become important to the model. Their physical location as they weave through the building now needs to be specifically described in the model along with the location inside the spaces of the network components they connect. This creates a more realistic view of the facility.

The electrical power for each of the components is now a factor as well. It is important to know how they receive electricity rather than simply if they do or do not have power. The main power supply for the facility comes in from an external source. There is also a back-up generator inside the facility. Both of these sources deliver power to a centralized distributor that routes it to electrical components. While this is an effective way to keep the facility as a whole from loosing power this means that only one power cable is running to the individual components. This is important in that individual components such as electric door locks or alarms, can be disconnected from power from inside the building without shutting down the entire facility.

Network components

For this scenario a more detailed description of the network is also required. The network must include the main server cluster, the secure room's server, the access terminal at the security desk, the alarms, and all the door locks. It must also include user permissions for multiple users to differentiate between a regular employee, a security officer, and admin access. The network is also connected to the Internet through a demilitarized zone (DMZ) and a firewall layer. This simulates a more realistic enterprise level network layout were computers inside the facility have restricted access to information depending on the user and allowing remote access to specified users.

Test scenario

In this scenario, the goal of the adversary is to gain access to the secure room server and take a hard drive holding critical information. I have particularly tailored the adversary towards this goal since an attack on such a facility would take premeditated thought and preparation to achieve.

Running this test scenario, the adversary is easily able to enter the facility from the outside with his employee credentials, bypassing the security measures of the perimeter fence and the door locks on the entryway and hallway doors. This is as far as his employee rights take him inside the facility. A real-world attacker would have a direct path in mind. He would know that the wiring for power and network connection runs through the ceiling of the server room, into the floor of the hallway, and into the wall and ceiling of the storage room so to power the door lock and give it access to the server to check the credentials of anyone who attempts access. Typically the wiring for a facility runs together for multiple spaces for organization, and to lower costs of installation during construction. Therefore the Ethernet and power cables that connect to the security desk, and secure room also travel into the storage room wall and ceiling. Even though the actor in the model is not scripted to directly attempt access into this space, the model will brute force all possible movements and eventually through lock-picking the door the maintenance room that shares a wall with the storage room.

Inside the maintenance room, the facility uses a drop ceiling to hide the infrastructure running over head such as wiring, air conditioning, and lighting. This is a common practice in many facilities since the drop ceiling is easy and inexpensive to install, and looks appealing. This means that if the adversary can reach the ceiling the has unrestricted access to open cabling. However, this is a difficult case to model. The height of the attacker is a factor along with their ingenuity in utilizing the items around him to reach the cables if he cannot on his own. For our purposes in the model this aspect is omitted on the grounds that the adversary would be able to climb or stack items from inside the room or would have known the height of the ceiling prior to the attack and brought a suitable tool to gain the needed height. With the cable now in hand the adversary can cut power or network access to any component down the line. Including the door lock in the secure room.

While this would seem to be a perfect case for the scenario there is a complication. Even though the door lock for the secure room is not engaged, should the door be opened the alarm would sound. Typically a security officer would have means to disarm the alarm. However, in this case the terminal at the security desk is locked or the security guard is stationed there, blocking use of his computer. This would spell the end of the potential attack. However, for this scenario the attacker has is remote accomplice. To connect remotely there would typically be a secure protocol such as a connection over SSL. In the test scenario, we injected the recent SSL heart-bleed bug into the network. Since the bug is rather recent and knowing that many companies are reluctant to patch since the patch must go through its own security audit, it likely that many facilities could still be vulnerable to this very attack. This is done by setting the network description to a version prior to the patch release. With this the remote accomplice could gain log in credentials to a much higher priority user such as the security officer. Armed with new credentials the remote user can log in as the security officer and disarm the alarm to the secure room.

The attacker can then move out of the maintenance room and back down the hallway to the secure room. There he does not even need to pick the physical lock since due to fire escape protocol the electric lock must fail open. With the alarm disarmed the attack now has access to his desired room. He can then locate and remove the proper hard-drive containing the critical information and leave the facility.

Results in Icarus

This scenario was placed into the Icarus tool unscripted. While I am looking for the specific vulnerability to be picked up, the order of events are not told to the model, only the capabilities of the actor and the details of the facility. While the original model ran nearly instantly, less than .01th of a second, the addition of more abilities and movement between spaces caused a significant increase in run time, up to more than 2 seconds. This result is not unanticipated, the level of increase is cause for concern. This model may be a much better example of a real-world facility it lacks the depth of a true case study. Since a real attacker would have many more skills and abilities along with a drastic increase in space the facility would occupy it would lead one to believe that a substantial increase in run-time would occur during a full facility. Another potential run-time problem occurred during the implementation of a remote accomplice. Since the accomplice's physical location does not have any effect on the model it was decided to implement him as an ability of the actor. Since the location of the actor can also have no bearing on the success or failure of the remote attack every time the actor either completed a task or entered a new space it would attempt this ability. This means that the state of the model's network was run through the MulVAL component, which drastically increased the running time. Also regardless of if the remote attack had been successful or not the model would still attempt this ability. Which means there were many unnecessary iterations of the MulVAL component. This was fixed by having a causal statement inside the actor object that was flagged if the remote attack had been successful. If the flag was set then the ability to attempt the remote vulnerability was removed from the actor. This means that the actor will nearly always attempt the attack first and in this scenario is successful immediately. However, if there needs to be a change in the state of the network before the remote attack can work, the model will attempt to try the remote attack after every action and cause a sever increase in run-time.

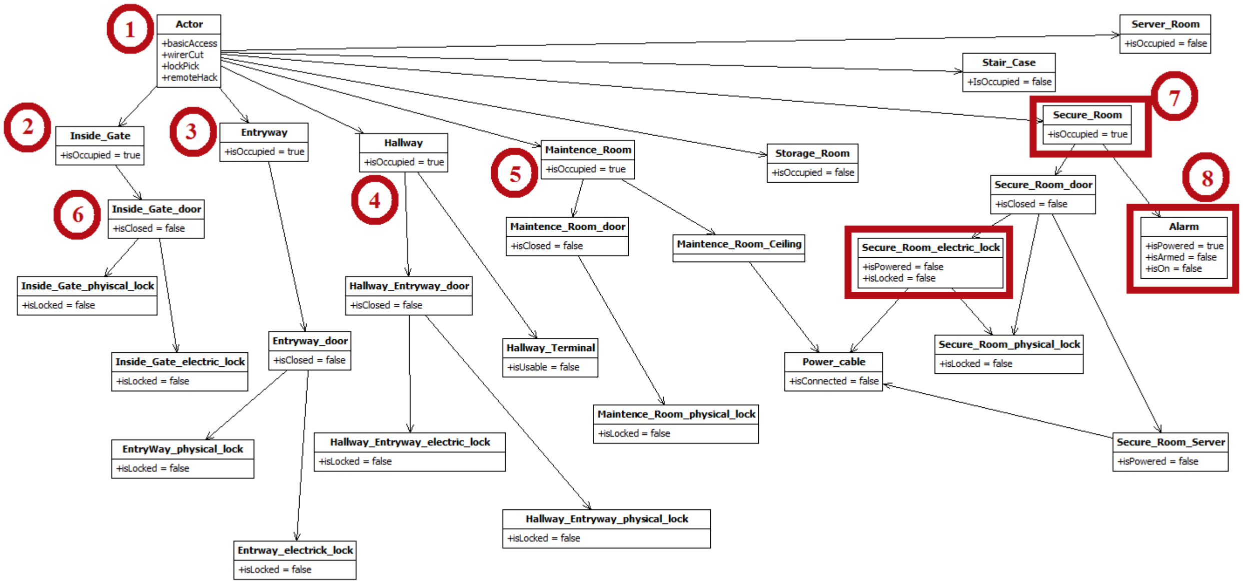


Figure 5.3 Counter-example generation of large scale model based off real-world facility

The Icarus tool did produce a counter-example consistent with the desired scenario, shown in figure 5.2, showing that the actor entered facility, picked a lock, cut power cables, implemented a remote vulnerability, accessed the secure room, and took a piece of hardware. However, the graphic display of the counter-example was incredibly cluttered and hard to read since there were so many interacting parts. This is problematic since the results are intended to be displayed to potentially non-sophisticated users. Looking at the graph, the actor (#1) is the adversary. The actor interacts with the rooms and the components inside to complete a goal. In 5.2 the actor is able to open the doors and occupy the spaces: outside the gate (#2), the entry way (#3), the hallway (#4) and the maintenance room (#5). The blocks labeled 2 through 5 represent the spaces inside the building. Connected to them are blocks such as #6 that represent components inside the room like doors or computer terminals. In figure 5.2 as the actor moves from room to room certain properties of components are changed that allow him access into the space. Eventually the actor is able to reach the secure room (#7 and highlighted in red). With the alarm (#8) and the lock to the door disabled, the actor is able to enter the space and renders the building vulnerable.

A curious result that occurred during the running of Icarus showed that the model tends to get stuck in repeating loops. The actor will enter a space, try all the possible abilities he has, then exit into the space he just came from. Then in the second space he will try all his abilities then return to the same space and repeat the process. This showed that rooms with only one entry and exit have to be labeled with no re-entry. This was corrected by limiting the number of times the actor could enter a space. This could lead to a problem where an attack could need the actor to return to a space he has already visited later on in the attack to complete and Icarus would not catch it.

5.4 Requirement analysis

The Icarus tool needed to provide the user a number of key characteristics in-order to be considered a useful tool. First and foremost it needed to be able to analyze the model and produce meaningful results. Secondly those results must be presented to the user in a way that both security specialists and non-specialist users can understand. Another major feature is that the tool details the steps necessary for the vulnerability to occur. Lastly since the tool is to be used not only by security experts but non-technical users as well the tool must be easy to use. The Icarus tool meets these conditions in the following ways.

Model analysis

The Icarus tool can take in a model in Bogor's augmented BIR language that contains a description of both the physical and cyber capabilities of a potential adversary along with a MulVAL description of the network setup and step through the rooms and cyber components of that facility to produce cyber-physical vulnerabilities. The model iterates through each of the potential actions the adversary is capable of in physical space until it reaches a cyber node, then the current state of the network is passed into the MulVAL component where the network integrity is checked. After MulVAL returns the state of the network, the model checking component continues forward until no new states can be reached or a security assertion is breached. If any of the assertions are broken the tool then declares there is a potential vulnerability in the system to the user.

Result presentation

If the Icarus tool only specified that a vulnerability existed, it would not be very useful to the facility. Therefore the result of the tool's execution needs to be more detailed than a binary yes or no answer. Instead the Icarus tool presents the user with a counter-example , see figure 5.1, which has been generated during execution that shows all the components involved in the vulnerability and the connections between them. This is a graphical interpretation that could be easily understood by both security experts and more business minded employees. In this manner the results of the model are visually displayed which gives them a more real-world meaning rather than a large block of text. This thesis believes that is a more appropriate representation that would have a larger impact for the facility described in the model.

Vulnerability fix extraction

A key component to the tool is the ability to not only find vulnerabilities but also allow the user to develop fixes from the results of the tool. While the tool itself does not show fixes automatically, the counter-example that is generated gives critical information that can be used to create fixes to specific vulnerabilities. Looking at the counter-example specific fixes can materialize as single points of failure inside the facility. If any component in counter-example graph that is in a acyclic path can be guarded so that is is removed from the graph and breaks the path, then the vulnerability would no longer be possible (at least not using that component). Looking at figure 5.2, there are no cyclic paths in the entire graph. This means that if we can remove any one of the components then that specific attack path will fail. Not all of the components can be secured. Therefore, looking for small components, such as the power line (#7 in figure 5.2), would create easier, more practical fixes.

Ease of use

A large theme in the entire project has been developing a useful tool for a large portion of potential users with varying levels of skill. For Icarus to be considered easy to use, it was determined that the tool should present its execution in a graphic user interface. With this portal for the tool this thesis believes that a larger group of both skilled and unskilled users would be able to operate the tool correctly and achieve meaningful results. The visual representation of the setup and execution of the model in combination with the visual representation of the results gives this tool a distinct advantage for usability.

Chapter 6 - Conclusion

In this thesis a new tool, Icarus, was developed to test large scale enterprise level network systems and the facility that the network and its components are housed in for combination cyber-physical attacks. The tool, Icarus, was built to be easy for non-technical or security conscious users to gain meaningful results out of as well as give detailed enough reports for users to understand the steps necessary for the vulnerability to occur. To achieve these results a model checking software, Bogor , was combined with a network vulnerability analysis software, MulVAL. The resulting tool was able to take in a model of the physical layout of the build and physical properties of the components within along with the electronic components of the encased network and a description of the network's setup to automatically analyze these pieces for combination cyber-physical attacks.

While Icarus preforms the basic necessary functions it was set out to do, the tool has yet to realize its full potential. There are a number of further improvements that would help to develop its capabilities and usefulness for facilities.

The most noticeable improvement would be a more descriptive modeling language. The Icarus tool uses a very customizable language, the Bandera Intermediate Representation (BIR). This language is very similar to a modern coding language in that it is extendable into custom objects. While this language is much better at allowing the model to describe the layout of the facility and the components of the network it has a difficult time describing the physical attributes of objects. The best case scenario is a giant white list of properties that an object in the real world has. This is problematic in that the effectiveness of the tool comes down to the thoroughness of the person writing the model. It is nearly impossible to know what properties of any single component even as simple as a door may or may-not be important during a break in or how those properties interact with the adversary. For this reason a simple white list is not sufficient to truly detail the object in the model.

The descriptiveness of the language does not stop at describing components of the facility. It becomes even more important when describing the actor attempting to breach the building. Explaining to the model the capabilities of a person is extremely difficult for many of the same reasons as describing an object. Again Icarus uses a large white list of capabilities to describe the actions an adversary might take. Just like before it is nearly impossible to know all the different things that a person can do and how any of those actions might be important to the model. There is also no generic model for what a person is, no way of giving a base line for a person to the model. Therefore every adversary must be run through the tool independently. This makes the tool much less useful since it would be so time consuming and may miss key adversary capabilities that could lead to a vulnerability.

References

[Cla08] E. M. Clarke. *The Birth of Model Checking*, 2008.https://www7.in.tum.de/um/25/pdf/Clarke.pdf

[McM92] K. McMillan. *Symbolic model checking: An approach to the state explosion problem.* 1992 http://www.kenmcmil.com/pubs/thesis.pdf

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y Zhu. *Symbolic Model Checking withoutt BDDs.* http://repository.cmu.edu/cgi/viewcontent.cgi?article=1426&context=compsci

[Bal88] R. Baldwin. *Rule based analysis of computer security*. Technical Report TR-401 MIT LCS Lab, 1988. http://194.47.250.18/pub/bitsavers.org/pdf/mit/lcs/tr/MIT-LCS-TR-401.pdf

[ZL96] D. Zerkle and K. Levitt. *NetKuang-A multi-host configuration vulnerability checker.* In *Proc. Of the 6th USENEX Seurity Symposium,* San Jose, California, 1996.

[FS91] D. Farmer and E. Spafford. *The cops security checker system*. Technical Report CSD-TR-993, Purdue University, 1991.

[TL00] S. Templeton and K. Levitt. *A requires/provides model for computer attacks.* In *Proceedings of the 2000 workshops on New security paradigms*. ACM Press, 2000.

[NSP03] The National Strategy for the Physical Protection of Critical Infrastructures and Key Assets, U.S. Dept of Homeland Security, February2003. http://www.au.af.mil/au/awc/awcgate/whitehouse/physical_strategy.pdf

[NSS03] The National Strategy to Secure Cyberspace, U.S. Dept of Homeland Security, February2003. http://www.au.af.mil/au/awc/awcgate/whitehouse/cyberstrategy.pdf

[CER14] CERT, Software Engineering Institute at Carnegie Mellon University, 2014. http://www.cert.org/vulnerability-analysis/

[ALL14] Alloy: a language & tool for relational models, MIT , 2014. http://alloy.mit.edu/alloy/index.html

[COM05] L. Compagna *SAT -Based Model-Checking of Security Protocols* , University of Edinburgh, 2005.

[EME08] E. Emerson *The Beginning of Model Checking*, University of Texas at Austin, 2008. https://www7.in.tum.de/um/25/pdf/Emerson.pdf

[GEA08] GEAR Concepts, 2008. http://jabc.cs.tu-dortmund.de/manual/index.php/GEAR_Concepts

[BOG05] Bogor Software Model Checking Framework: User Manual, SanToS Labratory, 2005. http://bogor.projects.cis.ksu.edu/manual/

[OGA05] X. Ou, S. Govindavajhala, and A. Appel *MulVAL: A Logic-based Network Security Analyzer*, Princeton University, 2005. http://people.cis.ksu.edu/~xou/publications/mulval_sec05.pdf

[OU05] X. Ou, Princeton University, A Logic-Programming Approach To Network Security Analysis, Princeton University, 2005. ftp://ftp.cs.princeton.edu/techreports/2005/735.pdf

[DAT05] Data Center Standers Overview, TIA-942, 2005. http://www.accu-tech.com/Portals/54495/docs/102264ae.pdf

Appendix A

Actor abilities	Boolean statement in model
Employee access rights	boolean basicAccess := true;
Security officer access rights	boolean securityAccess := false;
Lock picking ability	boolean lockPick := true;
Wire cutting ability	boolean wireCut := true;
Card swipe ability (actor has a security card)	boolean cardSwipe := true;
Door hinge removal ability	boolean doorHingeRemoval := true;
Execute remote attack	boolean remoteHack := true;

Actor abilities used through various models

Object property	Boolean statement in model
(room) basic Access	boolean basicAccess := true;
(room) officer Access	boolean securityAccess := false;
(room) occupied	boolean isOccupied := false;
(door) closed	boolean isClosed := true;
(door) removable hinges	boolean removableHinges := false;
(physical lock) locked	boolean isLocked := true;
(physical lock) lock Pick	boolean pickable := true;
(electronic lock,server) powered	boolean isPowered := true;
(alarm) armed	boolean isArmed := true;
(alarm) on	boolean isOn := false;
(wire) connected	boolean isConnected := true;
(wire) cuttable	boolean isCuttable := true;
(computer terminal) usable	boolean isUsable := false;
(electric component) networked	Boolean isNetworked := true;

Properties held by objects in various models

Actor	Electronic Lock
Room	Power Supply
Wall	Wire
Floor	Computer Terminal
Ceiling	Server
Door	Alarm
Physical lock	

List of Objects Modeled

```
record Object {}
record Storage_Room extends Object {
        Wall Storage_Room_North_Wall;
        Wall Storage_Room_South_Wall;
        Wall Storage_Room_East_Wall;
        Wall Storage_Room_West_Wall;
        Door Storage_Room_Door;

        boolean isOccupied := false;
        boolean basicAccess := true;
        boolean securityAccess := false;
}
record Storage_Room_North_Wall extends Object {
        Wire Power_Cable;
}
record Storage_Room_Floor extends Object {
}
record Storage_Room_Ceiling extends Object {
        Wire Power_cable;
}
record Storage_Room_Door extends Object {
        Physical_Lock Storage_Room_Lock;
        boolean isClosed := true;
        boolean removableHinges := false;
}
record Storage_Room_Lock extends Object {
        boolean isLocked := true;
        boolean pickable := true;
}
record Secure_Room_Electric_Lock extends Object {
        boolean isPowered := true;
        boolean isLocked := true;
        boolean isNetworked := true;
}
record Main_Power_Supply extends Object {
        boolean isPowered := true;
}
record Power_Cable extends Object {
        boolean isConnected := true;
        boolean isCuttable := true;
}
record Hallway_Terminal extends Object {
        boolean isUsable := false;
        boolean isPowered := true;
        boolean isNetworked := true;
}
```

Example of objects in various models