Simulating the Effects of Releasing Malware into the Internet of Things

Jamie Knowles

School of Computer Science and Informatics

Cardiff University

A thesis submitted as part of the requirement for the degree of

Bachelor of Science in Computer Science

May 11, 2018

Acknowledgements

Firstly, I would like to thank my family; their unconditional love and support continually motivates me to be this best I can be everyday. Secondly, I would like to express my appreciation to my supervisor Eirini Anthi for her support throughout the course of this project; a project that would not have been successful without her input and guidance. Furthermore, I would also like to thank all of the friends I have made in the last four years who have made my experience at University an absolute blast!

Abstract

The emergence of the Internet of Things has caused a variety of issues in the world of cyber security. Many device manufacturers are distributing insecure devices to customers and many customers are unaware of how to keep their devices secure. This has allowed the Internet of Things to become a goldmine for hackers and permitted them to complete their objectives with relative ease whether that be to take down critical infrastructure or steal sensitive information.

To further assist in raising awareness of the insecurities of the Internet of Things this project involves the development of software that simulates malware propagation on real device data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By the end of this year, there will be more than eight billion Internet of Things (IoT) devices connected worldwide[10]. Many of these devices are left vulnerable and are therefore constantly exploited by hackers across the globe. In October 2016, a company that controls a large majority of the Internet's domain name system (DNS) infrastructure named Dyn was hit by a distributed denial of service (DDoS) attack that led to numerous popular sites suffering downtime including Twitter, Netflix and Reddit. The attack was orchestrated by a strain of malware known as Mirai, the strain infects vulnerable IoT devices to form a botnet. In this instance, the botnet was used to coordinate a DDoS attack on Dyn. The attack is known as one of the largest disruptions the Internet has suffered [26]. It is clear that, with the amount of IoT devices rapidly rising, more needs to be done to raise awareness of how vulnerable the Internet of Things is and what the consequences of these vulnerabilities could be. Angel Martin del Rey, a Mathematics professor from the University of Salamanca believes that the fight against malware is missing one key ingredient, software simulating malware propagation[19].

Therefore, this project aims to create an application that is capable of simulating the effects of releasing malware into the Internet of Things. The system will give a visual representation to the user of devices becoming infected and a representation of the steps taken when attempting to infect a device. The system will demonstrate using these simulations and visulations how vulnerable the Internet of Things is and what the consequences of these vulnerabilities could be.

This project is intended to be built as a useful tool for IoT stakeholders who are unaware of the insecurities of the Internet of Things. Such examples of interested or concerned entities could be businesses selling devices or customers buying devices.

The scope of this project is to deliver a system that allows the simulation of multiple strains of malware on a given set of devices and visualises the results of a

simulation in a user-friendly fashion. To achieve this real device data will be required that will be used to simulate IoT devices, this will be acquired from Shodan (shodan.io). A simulation of an attempt to infect a set of devices with a given strain of malware can then be performed by using this data and information gathered about the malware being simulated. Although many strains of malware may propagate to other devices on a network once infected, this will not occur during the simulations that will be performed. The effects of the simulation will be shown through a graphical user interface. The project will require in-depth research to be conducted on the Internet of Things, existing IoT malware, probability theory and the technologies at use in the project where proficiency may be lacking.

The system will be designed with modularity in mind so that each service is lightweight and has its own unique, well-defined goal. Although some tasks will require the project to be executed in a sequential manner, the phases within the stage of development will require a more flexible approach due to the modularity of the system. Therefore, the Scrum methodology will be used which involves working towards sub-goals within specific time frames, called sprints. Time will be taken each day to reflect on previous work, decide on future work and how any problems that have occurred may be solved. It is important to note that Scrum is usually ideal for small teams but utilising the key principles of this methodology will allow continuous improvement through constant reflection of progression. During the duration of this project Scrum will be executed in one week sprints.

It is assumed that users of the system will have a sufficient understanding of what an IoT device is and what it means to infect a device with malware. The users should also be able to run JavaScript within their browser and have an active Internet connection unless the system is run locally.

Chapter 2

Background

2.1 The Internet of Things (IoT)

2.1.1 An Introduction to IoT

The Internet of Things is the network of all physical devices embedded with electronics, software, sensors, actuators and connectivity which enables them to connect and exchange data. Each device is able to operate within the existing Internet infrastructure. As previously mentioned, by the end of this year there will be more than eight billion IoT devices connected worldwide. By 2020, experts estimate that IoT will consist of more than 20 billion devices and spending on hardware from both consumers and businesses will reach almost $3 trillion[10].

2.1.2 An Anatomy of an IoT Device

An IoT device is a special-purpose device, that connects wirelessly to a network and transmits and receives data in order to monitor or control a "thing"[17].

Sensors and Actuators

In order for an IoT device to work it is reliant on sensors, actuators or both. The sensors job is to acquire data and the actuators job is to control the data or act on the data. For example, a sensor may monitor and provide data about the temperature and then an actuator may adapt the controls in a smart thermostat accordingly[17].

Firmware

IoT devices also require firmware which can either be embedded or operating system-based. Embedded firmware is specialised software that runs on the device and interacts with the devices hardware. With the necessity for more sensors, greater data

processing, storage capabilities and more - the demand for more complicated software to manage and exploit the new capabilities has also grown. This is where operating system-based firmware is used, this type of firmware provides a layer of abstraction between the hardware and other software that runs on the device. This enables a familiar division of labour for businesses; embedded software engineers can now spend their time writing device drivers and application programmers can spend their time writing the software that makes the device "smart". A popular operating system for many IoT device manufacturers is BusyBox (busybox.net), a stripped down version of the Unix operating system that contains many of the most common utilities, has a very small footprint, and provides many capabilities of Unix in a single executable[17].

2.1.3 IoT; Risk or Reward?

There is no question that the Internet of Things is transforming the world in a multitude of ways. However, security has not always been emphasised by manufacturers during the stage of product design. This, and the fact that most consumers are not aware of what they can personally do to keep their devices secure means that IoT can also be highly disruptive. The Dyn attack discussed in chapter 1 is a good example of IoT being exploited to cause mayhem. Do examples like this make IoT too big of a risk? Now that IoT devices are being used in delicate fields such as health care[15] it is only a matter of time before an IoT-based cyber attack is executed that could result in causing physical harm to an individual. Therefore, if IoT security is not improved the risks will eventually outweigh the rewards.

2.2 The Rise of the Cyber Threat

2.2.1 An Introduction to Cyber Attacks

A cyber attack is an attempt by hackers to damage or destroy a computer network or system. Over the years, cyber attacks have been extremely costly to businesses. When Sony's PlayStation Network was attacked in 2011 it suffered an outage of 23 days, Sony was pursued with all sorts of legal action and it was believed that over 100 million people were thought to have been affected by the attack. Sony executives suggested that the breach would cost the company $171 million[11]. In recent years, more state-sponsored cyber attacks have occurred. These attacks have heightened tensions between various countries. After an attack was performed on the Democratic National Committee (DNC) in 2015/2016, the CIA attributed Russia to

the attack and after further investigation concluded that they had intervened in the 2016 U.S. election[1] causing tensions between the United States and Russia to rise tremendously.

2.2.2 Who is the Adversary?

Cyber attacks can be performed by anyone with a reasonable computing knowledge and access to the Internet these days. However, the more damaging and sophisticated attacks are usually performed by either Black Hat cyber security experts or state-sponsored hackers. As previously mentioned, the number of state-sponsored attacks have risen with various states such as Russia and North Korea making big noise. In 2017, the Foreign Office Minister for Cyber Security, Lord Ahmad attributed the WannaCry ransomware incident that impacted 300,000 computers in 150 countries including 48 NHS trusts to North Korean actors, the Lazarus group[3].

2.2.3 A Brief History of Cyber Attacks

In recent years, cyber attacks have caused disruption across the globe. This section details a brief history of some of the most famous cyber attacks that caught the media's attention.

Year	Summary of Cyber Attack
2010	The Stuxnet worm infiltrates a nuclear factory. It is reported that Iran decommissioned around 20 per cent of its centrifuges in the Natanz plant during the attack[5].
2011	Sony's PlayStation Network was attacked in 2011. It suffered an outage of 23 days, Sony was pursued with all sorts of legal action and it was believed that over 100 million people were thought to have been affected by the attack. Sony executives suggested that the breach would cost the company $171 million[11].
2014	Sony Pictures is hit by an attack attributed to North Korea due to its upcoming release of satirical comedy The Interview, which involves a plot to assassinate North Korean leader Kim Jong-un. The attack exposed embarrassing emails and personal details about some of the world's biggest movie stars[4].
2016	A Democratic National Committee (DNC) computer is compromised by Russian hackers and Wikileaks publishes Hillary Clinton's emails obtained from the hack[8].
2016	Domain name system (DNS) infrastructure company, Dyn, is hit by a distributed denial of service (DDoS) attack causing numerous popular sites including Twitter, Netflix and Reddit to suffer downtime[26].
2017	WannaCry ransomware impacts 300,000 computers in 150 countries including 48 NHS trusts[3].
2017	Petya ransomware spreads through large firms including the advertiser WPP, food company Mondelez, legal firm DLA Piper and Danish shipping and transport firm Maersk, leading to PCs and data being locked up and held for ransom[22].

2017	Equifax is hacked and 146.6 million names, 146.6 million dates of birth, 145.5 million social security numbers, 99 million addresses and 209,000 payment cards are exposed[7].

Table 2.2: A brief history of cyber attacks

2.3 Malware, the Ultimate Cyber Weapon

Malware, short for malicious software, is a term used to refer to any software designed to cause damage to a single computer, server, or computer network. A strain of malware can come in various forms whether it be spyware, ransomware or just a regular virus[14].

2.3.1 The Cyber Kill Chain

A cyber kill chain maps the stages of a potential security breach. It can be used to help us understand the process malware or an actor takes when performing a cyber attack. The following cyber kill chain shown in figure 2.1 was derived by Lockheed Martin (`lockheedmartin.com`) from a military model[13].

Figure 2.1: The cyber kill chain

1. Reconnaissance.
   Reconnaisance is the information gathering stage. This is where an attacker

seeks information that might reveal vulnerabilities of a system such as open ports or an outdated operating system.

2. Intrusion.
   Intrusion involves using the information gathered from the reconnaissance stage to gain access to a system. This could be by trying to brute-force a weak open port such as Telnet.

3. Exploitation.
   Once access is gained to the system it can then be exploited by performing tasks such as installing tools.

4. Privilege Escalation.
   Escalating privileges is the next stage. This is done because having access to a system is great, but having full access to a system is better.

5. Lateral Movement.
   In order to gain more access or find more information the attacker may need to move around a network from system to system.

6. Obfuscation (anti-forensics).
   Obfuscation is the task of hiding an attackers tracks to mislead the owner or forensic analysts, this can be done by performing such tasks like editing log files or deleting files.

7. Denial of Service.
   A denial of service (DoS) attack may be performed on a network or infrastructure to disrupt access, crash systems and flood services.

8. Exfiltration.
   This stage involves copying, transferring or moving data to a controlled location. This data can then be used by the attacker. For example, the attacker may choose to leak the data to the public.

2.3.2 An Anatomy of IoT Malware

The Attack Vector

To perform an attack, the attacker needs to hit an attack surface, an attack surface is the sum of all of the target devices vulnerabilities. Once an attacker has become familiar with an attack surface they can identify an attack vector, this is a point

of entry that can be exploited to gain access to a device[17]. An IoT device attack surface can contain an abundance of attack vectors but here are some of the most common:

- Weak passwords
- Lack of encryption
- Backdoors
- Internet exposure

Additionally, the Open Web Application Security Project (OWASP) (owasp.org) runs the IoT Attack Surface Area Project, a project that maintains a list of potential vulnerabilities in the IoT attack surface.

The Attack

An attack comes in two phases; the scan and takeover phase and the attack launch phase. These phases are usually executed by a Command and Control (C2) server[17].

1. Scan and Takeover.
   The C2 server finds vulnerable hosts and attempts to gain access. If access is gained to a host then the the host, attack vector and any other essential information is reported to the C2 server. The C2 server then pushes the malware to the device and that is necessary to perform the attack. The host is now under control by the C2 server and awaits further instructions.
2. Attack Launch.
   The C2 server can now launch an attack, this could be to form a botnet and perform a DDoS (distributed denial of service) attack or alternatively sensitive data could be retrieved from a controlled device.

2.3.3 Countermeasures

In truth, malware is always adapting and finding new ways to infect targets and so there are no countermeasures that can provide complete security. However, the following countermeasures provided by IBM[17] can give additional security and make IoT devices less vulnerable.

- Always change default passwords

- Remove devices with Telnet backdoors
- Never expose a device directly to the internet
- Run port scans on all of your machines and close ports accordingly

2.3.4 Famous IoT Malware

2.3.4.1 Mirai

Mirai was the strain of malware used to perform the Dyn attack discussed in chapter 1. It works by scanning the internet for hosts with an open Telnet port (TCP port 23). Once a host is identified it uses a list of default usernames and passwords to gain access to devices that are running BusyBox. Once access has been gained, the malware is installed and a Command and Control (C2) server is made aware. It then awaits further instructions. The C2 server can then be used to launch a flood of various kinds of traffic, overwhelming a chosen target host. Mirai mostly used infected CCTV camera devices to carry out the Dyn attack. During Mirai's scan for eligible hosts, the malware also checks a "do not mess with" list of servers that include General Electric, Hewlett Packard, and the U.S. Department of Defense. If the host is on the list, it is not targeted[17].

2.3.4.2 Brickerbot

Brickerbot is another BusyBox-based strain of malware, it differs from Mirai by bricking target devices. Making them unusable. Brickerbot does this through a series of BusyBox commands that wipe everything from the devices internal storage through the Unix "rm" command, along with commands that reconfigure the kernel, and finally reboot the (now useless) device[17].

2.3.4.3 BASHLITE

BASHLITE is another strain of malware that infects BusyBox-based systems. The original version in 2014 exploited a vulnerability in the bash shell, this vulnerability was named "Shellshock" and it exploits devices to form a botnet. BASHLITE is considered the precursor to Mirai. In 2016 it was reported that one million devices had been infected. 96 percent of these devices were identified as being IoT of which 95 percent were cameras and DVRs with 4 percent being home routers[23].

Chapter 3

Specification and Design

3.1 Overview

The project described throughout this report has one paramount goal; provide users with a greater understanding of how vulnerable the Internet of Things truly is. This goal will be achieved through a web application that will allow users to visualise a simulation of malware being released on a set of real IoT devices. This chapter aims to explain the requirements necessary to fulfil this goal as well as give readers an insight into design decisions made as well as the choice of tools, technologies and resources.

3.2 Requirements

The following section describes all of the functional and non-functional requirements that are essential and desirable for the system to be built.

Essential

Functional

1. Display devices on a map	
Acceptance Criteria	Each device has a visual representation of itself on a map that has been placed in the given location of the device.
Justification	This will introduce the ability to be able to visually represent each devices reactions to the malware being released.

2. Display specific device information	
Acceptance Criteria	When a device marker has been clicked on, information about the device is such as its IP address and operating system is displayed.
Justification	This gives users a more detailed view of a device to view the characteristics of a device that will be used by the simulator to decide if the infection of the device is successful.

3. Simulate the effects of releasing a specified strain of malware	
Acceptance Criteria	The user is able to selectively release a strain of malware over a set of devices. The strain selected is released over each device and feedback powered by the device data is given both graphically and in a more detailed text view. Once a simulation is complete the results are then shown to the user.
Justification	This requirement is the projects main goal, the result of fulfilling this requirement will allow the user to perform simulations that will demonstrate how vulnerable the Internet of Things is.

4. Display the state of devices in a meaningful way	
Acceptance Criteria	When a simulation is taking place, the user should be able to graphically view the state of devices in a way that intuitively describes the state of the device.
Justification	This will let the user perceive infected devices.

5. A strain of malware should only be attempted to be released on a device that meets the malware's target device specification	
Acceptance Criteria	A strain of malware should have a target device specification that gives information on what sort of device it can infect. When a simulation is executed, the simulation will compare the malware's target specification to the specification of the device being released on.
Justification	This will allow the malware to only be released on devices that the malware can run on, improving the simulation's accuracy.

6. A strain of malware should only gain access to a device the device has a high chance of being vulnerable to an attack vector that the malware is able to pursue	
Acceptance Criteria	The chance of a pursued attack vector being successful is determined by the device data provided and statistical findings about the attack vector.
Justification	This will allow the malware to only gain access to devices that are vulnerable to the attack vectors exploited by the malware, improving the simulation's accuracy.

7. A relatively short description should be provided for each strain strain of malware that can be simulated	
Acceptance Criteria	A page giving a short but informative technical description of each strain of malware that can be simulated on is provided.
Justification	This provides the user with a more detailed view of the malware they are releasing, including information about how the malware propagates.

Non-functional

1. The user interface should have a high level of usability and accessibility	
Acceptance Criteria	The user interface follows Google's Material Design guidelines[12] and meets Nielsen's usability heuristics[16].
Justification	The user interface will have high usability and a modern elegant design, enriching the users experience.

2. The simulator should allow a large amount of devices to be simulated on with good performance	
Acceptance Criteria	The simulator can perform a simulation on >5000 devices without showing performance issues.
Justification	A user may want to simulate on a large amount of devices.

Desirable

Functional

1. A user should be able to specify a location in which to perform an attack	
Acceptance Criteria	The user can select a country in which to perform an attack.
Justification	This will allow users to analyse the vulnerability of devices within a specific location.

2. The user should be able to pause, play and stop simulations	
Acceptance Criteria	The user is able to pause, play and stop simulations through the use of buttons.
Justification	This will provide the user with the option of looking at the "state of play" at a specific point in time, or if the user has incorrectly started a simulation they can stop the simulation instead of reloading the page.

3. The user should be able to choose the speed of the simulation	
Acceptance Criteria	The user is able to select a simulation speed from the following options: Normal, 2x, 3x, 5x, 10x, 20x.
Justification	If a simulation may be projected to take a long time, the user can choose to speed up the simulation to their level of choice.

4. The user should be able to add custom malware	
Acceptance Criteria	The user can add a custom strain of malware by giving itss name, the specification of device it targets, the attack vectors it exploits to gain access to a device and the actions it performs after infecting a device.
Justification	This will allow users to simulate malware against a set of devices that carries characteristics that may not have been seen before.

5. The user should be able to add custom devices	
Acceptance Criteria	A user can add new devices to the current dataset and then perform simulations against the new dataset of devices.
Justification	This can allow users to simulate malware against devices that may be affiliated with them such as their home network in some way to see how vulnerable they are.

6. The user should be able to switch between a light and dark user interface	
Acceptance Criteria	The user can switch between a light and dark colour scheme at the click of a button. The colour scheme selection should stay the same when the page is reloaded.
Justification	This allows the user to select a colour scheme they deem to be better from a visual and accessibility point of view.

Non-functional

1. Each component should be run in an appropriate cloud computing service.	
Acceptance Criteria	All client-side and server-side components are run in appropriate services provided by a reliable cloud computing company such as Amazon Web Services (AWS) or Google Cloud.
Justification	Running each component in a cloud computing service can help provide high availability, scalability and reliability.

3.3 Design

3.3.1 System Architecture

The application developed as part of this project uses the Model-View-Controller (MVC) architectural design pattern. This is a commonly used design pattern for web applications that promotes parallel development and code reuse by decoupling three major system components; the model, the view and the controller.

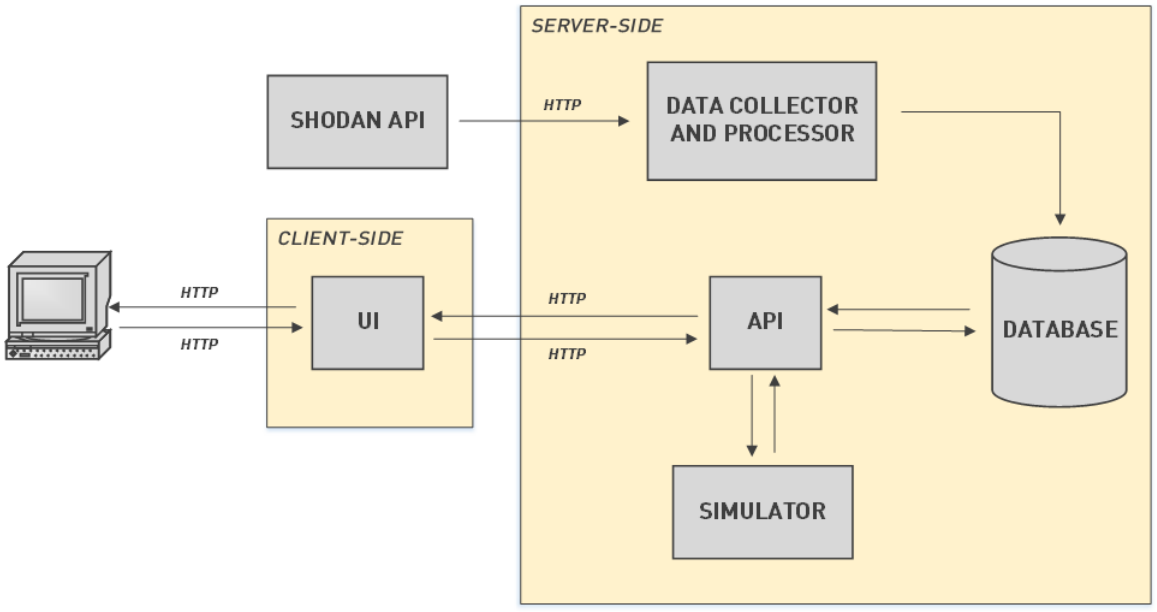

Figure 3.1: A diagram of the systems architecture

Figure 3.1 describes the systems architecture. This architecture utilises the MVC design pattern by using the user interface (UI) as the view component of the system. The UI then interacts with a RESTful API by sending requests over HTTP, the API acts as the controller component of the system. The API will then process these requests by manipulating the model component using business logic. The controller

can then respond to the these requests in the correct manner and the UI will render the updated output. In figure 3.1 there are also three other major components of the system; the simulator, the data collector/processor and the database. The data collection and processing module collects and processes the device data and then stores it within the database. Whenever a request to release a specific strain of malware is triggered, the API receives the request from the UI and then offloads the request to the simulator, the simulator contains a knowledge base of different strains of malware and using this knowledge base the simulator executes a simulation of the strain of malware being released on the current set of devices within the given criteria. The simulator creates a simulation object and appends events until the simulation is complete. The simulation object is then sent back to the user interface. The user interface then "plays" the simulation by iterating over the events in the simulation object. Furthermore, this modular approach allows each service to be lightweight and have it's own unique, well-defined goal thus simplifying the understanding of the system and easing maintainability.

3.3.2 User Interface

This section outlines the decisions made when designing the user interface. When designing the user interface Google's Material Design guidelines, a "visual language that synthesizes the classic principles of good design with the innovation and possibility of technology and science" [12] were followed. Further to following Material Designs guidelines, this also meant utilising the components and tooling available on their website (`https://material.io`). Furthermore, to ensure usability Nielsen's usability heuristics[16] were also followed.

Colour Schemes

In order to satisfy the desirable user requirement of the user being "able to switch between a light and dark user interface", it is necessary to develop two colour schemes. One light, and one dark. The dark colour scheme can be seen in figure 3.2 and the light colour scheme in 3.3.

Figure 3.2: The dark colour scheme

Figure 3.3: The light colour scheme

Fonts

The fonts shown in figures 3.4 and 3.5 will be used throughout the user interface, Roboto is the standard typeface on Android and was designed by Google with Material Design in mind. Roboto Mono is the monospaced addition to the Roboto family, using a monospaced font would add more of a "command-line vibe" to various aspects of the system such as the simulation log history.

Maps

I decided to use a map as a means of visualising the devices because this would allow users to not only analyse how vulnerable IoT devices are but also compare how

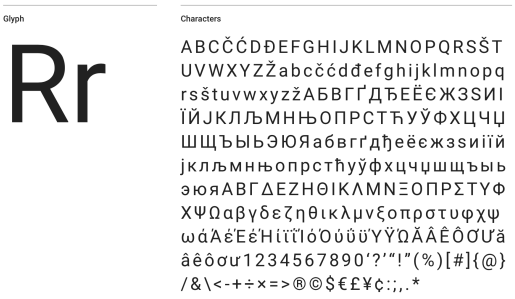

Figure 3.4: Roboto font

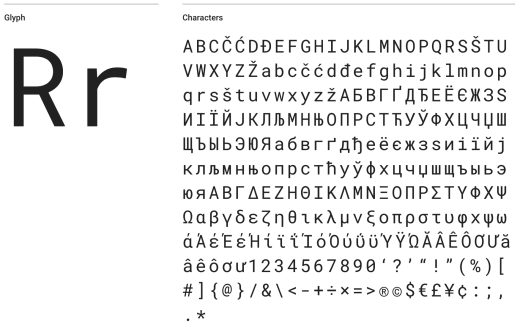

Figure 3.5: Roboto Mono font

vulnerable they are in specific locations. The following maps could all potentially be used as a means of visualising the map to display the devices on. The maps that will be used in coordination with the dark colour scheme (see figure 3.2) can be seen in figures 3.6 and 3.7. The maps that will be used in coordination with the light colour scheme (see figure 3.3) can be seen in figures 3.8 and 3.9. They were all created by CartoDB[6].

Figure 3.6: DarkMatter map without labels

Figure 3.7: DarkMatter map with labels

Figure 3.8: Positron map without labels

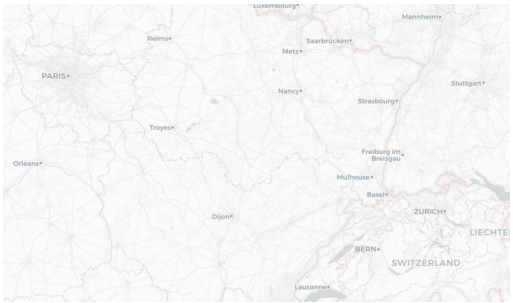

Figure 3.9: Positron map with labels

Artboards

The following artboards shown in figures 3.10, 3.11, 3.12, 3.13 and 3.14 use Material Design components, the dark colour scheme shown in figure 3.2 and the dark map shown in figure 3.6. They are how the user interface is portrayed to look from certain aspects.

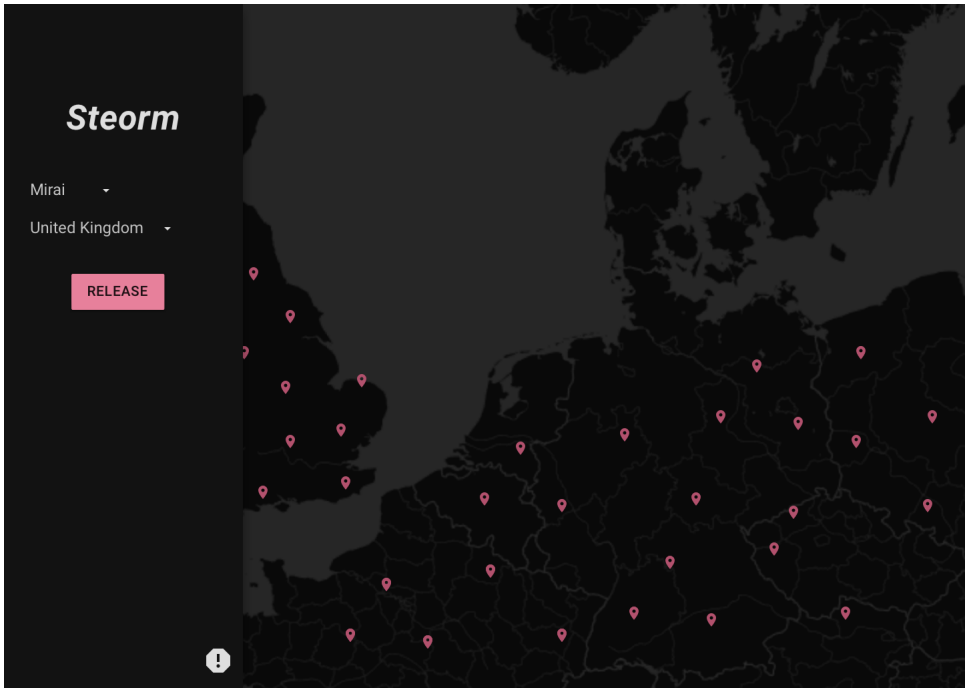

Figure 3.10: System state before any malware has been released

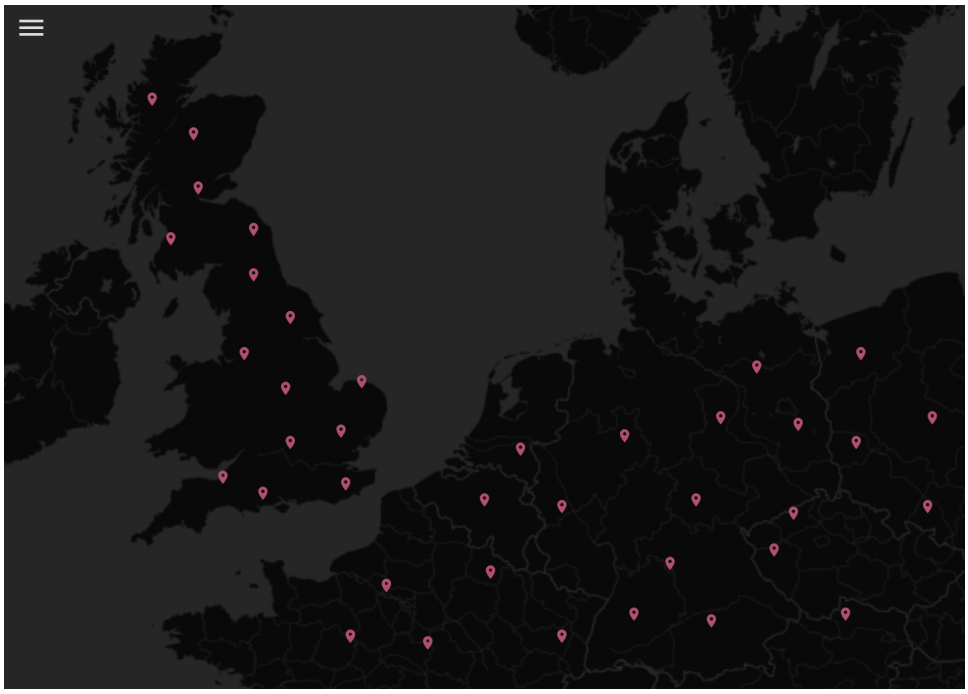

Figure 3.11: System state before any malware has been released with side-menu closed

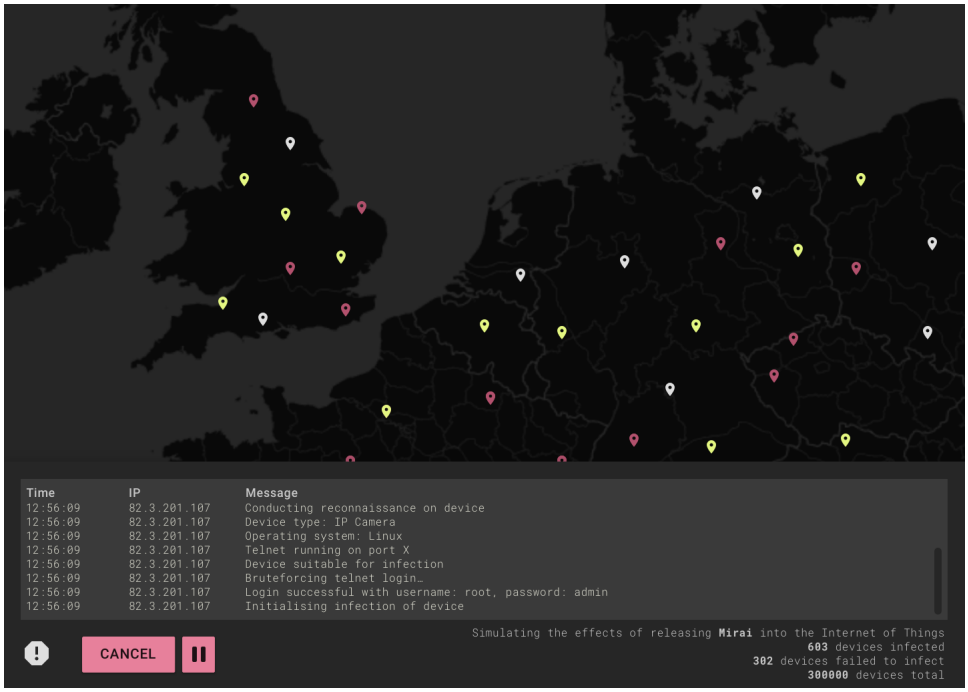

Figure 3.12: During a simulation taking place

Figure 3.13: After a simulation has taken place

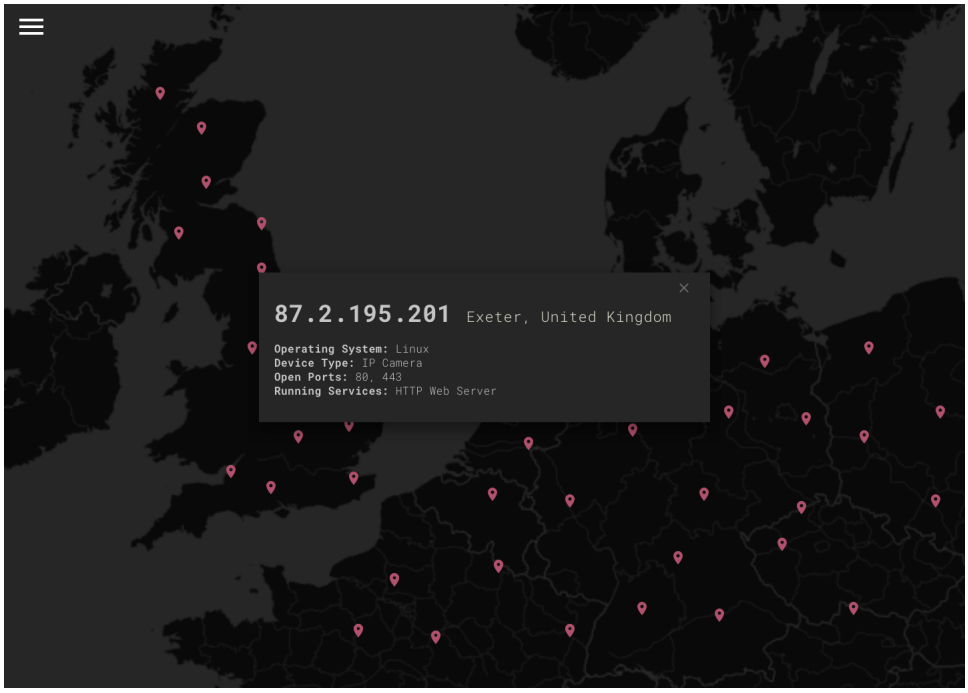

Figure 3.14: Device information displayed from clicking on a device marker

3.3.3 Simulator

This section explains the kill chain of events and underlying mathematics necessary to perform a simulation.

An Adapted Kill Chain

Using the modelled cyber kill chain shown in figure 2.1; a new kill chain inspired by the previous can be built that contains the correct steps necessary to performing a simulation.

1. *Reconnaissance* - Reconnaissance involves observing a device to discover characteristics associated with the device such as its operating system and public-facing ports. There are various means of performing reconnaissance actively and passively such as by conducting port scans. Reconnaissance is important as it allows a strain of malware to determine if a device can be infected. The data that Shodan will provide contains all the information necessary to determine if a device can be infected and therefore will allow us to simulate the reconnaissance aspect of malware propagation. Therefore, this stage is used to compare the specification of a device that the malware runs against and the specification of the device the malware is attempting to infect.

2. *Intrusion* - At this stage a target device has been identified and so an attempt to gain access to the device is made using the released malware's set of attack vectors that it pursues. We cannot know for certain if a device is accessible without actually trying to gain access to the device therefore it will be necessary to come up with the mathematical backing that predicts the probability of a device being accessible.

3. *Exploitation* - Once access has been gained to a device, the exploitation stage then occurs. This, simply put, is where the chosen strain of malware performs its specific method/s to infect the target device. For example, as discussed in section 2.3.4.1, Mirai infects devices by exploiting BusyBox commands allowing the malware to gain control of a device. However, if this stage is reached it is already proven that the target device is vulnerable and therefore abstracting the steps taken during this stage down to the one step of "infecting device" rather than the specific steps a strain of malware may perform is sufficient.

4. *Action* - This stage involves the malware performing its purpose, such as data exfiltration, denial of service, or encryption for ransom. This stages essentially combines stages 4-8 shown in figure 2.1. However, for some malware their purpose is arbitrary and in those cases an abstract "action" will be provided. For example, malware such as Mirai could be used to perform DDoS attacks but that is not certain therefore Mirai's action will be the following - "device under control by command and control (C2) server". However, malware like Brickerbot have a defined purpose, in Brickerbot's case its action would be the following - "device bricked".

Markov Chain

A Markov chain is "a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event"[9]. This makes it an ideal candidate to encode our stages of the malware kill chain shown above in section 3.3.3. It will also be necessary to have absorbing states, these are states which once entered, cannot be left. These purpose of these absorbing states is to end the current simulation on a device. In 3.15, each state in the Markov chain can be seen as well as the absorbing states.

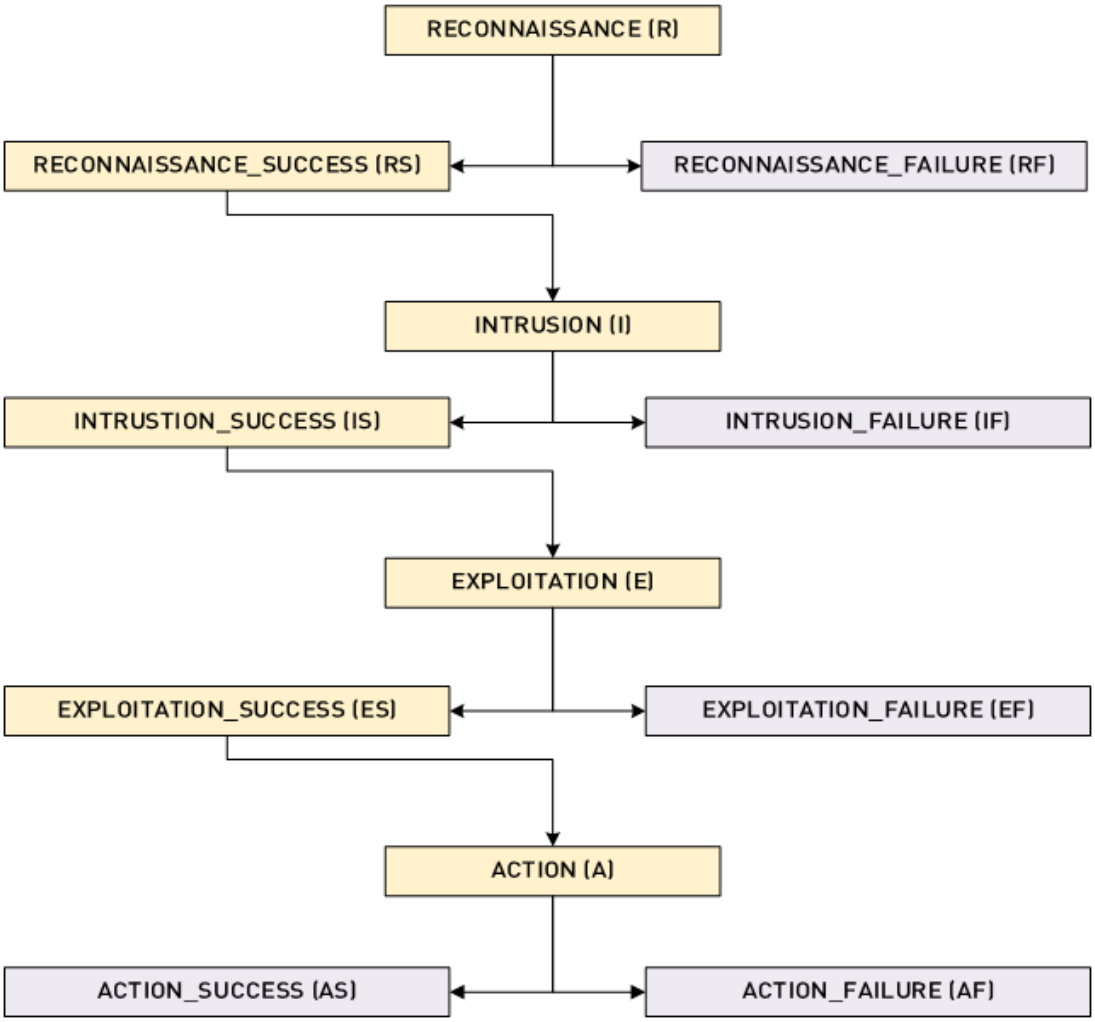

Figure 3.15: Markov chain for the malware kill chain

In a real-world scenario, events such as losing connection to the targeted device or being caught by an Intrusion Detection System (IDS) could occur at various states. However, an assumption can be made that events such as these do not occur. Another assumption that can be made is that the exploitation and action stages always result in success. This decision can be made because although the results given may not be completely accurate with real-world scenarios, the simulation will still allow users to see the vulnerabilities of IoT devices which is the goal of the project. This is because an IoT device can still be considered vulnerable if the intrusion stage has resulted in success. Further work may result in doing research and finding data to support the possibility of exploitation and action resulting in failure. The table shown in 3.1 details the transition matrix for the Markov chain showing the probability of moving from one state to another where each cell is $P(COL|ROW)$. Some of the transitions,

denoted *, are determined dynamically depending on the data provided.

	R	RS	RF	I	IS	IF	E	ES	EF	A	AS	AF
R	0	*	$1 - P(RS\|R)$	0	0	0	0	0	0	0	0	0
RS	0	0	0	1	0	0	0	0	0	0	0	0
RF	0	0	1	0	0	0	0	0	0	0	0	0
I	0	0	0	0	*	$1 - P(ES\|E)$	0	0	0	0	0	0
IS	0	0	0	0	0	0	1	0	0	0	0	0
IF	0	0	0	0	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0	0	0
ES	0	0	0	0	0	0	0	0	0	1	0	0
EF	0	0	0	0	0	0	0	0	1	0	0	0
A	0	0	0	0	0	0	0	0	0	0	1	0
AS	0	0	0	0	0	0	0	0	0	0	1	0
AF	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.1: Markov chain transition matrix

Predicting Kill Chain Stage Times

The aim of this project is to demonstrate the vulnerabilities of IoT devices but as this is being done through a simulation a sense of realism is still necessary, one part of this includes predicting how long each stage of the kill chain may take. However, as no data is available that can assist with these estimations it will be difficult to predict these time-frames therefore estimated time-frames have been supplied in the table below (see table 3.2). These estimations are purely based on personal expectations. Using personal estimations is not the best solution but this issue will not affect the projects goal of proving how vulnerable the Internet of Things is.

	Min	Max
Reconnaissance	0.1	2
Intrusion	2	5
Exploitation	5	10
Action	5	10

Table 3.2: Estimated timings for malware kill chain

3.4 Technologies

During this project a variety of technologies will be used. This section aims to provide brief descriptions of the main technologies utilised and a justification for their usage.

Programming Languages

Python

Python was chosen as a language of choice due to already having a high proficiency in the language, which was deemed an important characteristic when choosing technologies to use throughout due to the short time span of the project. Furthermore, multiple Python web frameworks exist that would be useful to develop the RESTful API necessary to accommodate conversations between the user interface and database as well as the user interface and simulator. All python code will be written using Python 3.

JavaScript

JavaScript was also chosen as a language of choice due to already having a high proficiency. Further to this reason, it is the most common language for building web applications, thus JavaScript and JavaScript-based libraries will be used to develop the user interface component of the application.

Libraries and Frameworks

React

React (`reactjs.org`) is a JavaScript library that will provide a means of creating the user interface component of the application with fast render speed. It is one of the most common libraries for developing user interfaces and also offers *createreactapp*, removing the need to configure tools such as Webpack or Babel.

Material-UI

Material-UI is a library containing React components that follow Google's Material Design. It contains all of the necessary Material Design components such as Drawer, SelectField and RaisedButton that would be necessary to implement the designs as shown in 3.3.2.

Leaflet

Leaflet (leafletjs.com) is an extremely lightweight JavaScript library for interactive maps. It provides all of the features necessary for the mapping aspect of the user interface. Paul Le Cam et al have also built a library that offers Leaflet features as React components (github.com/PaulLeCam/react-leaflet), which is an ideal solution for integrating the two libraries. Leaflet can also scale, which is necessary in the case of this project as lots of markers would potentially need to be placed.

Flask

Flask is a web framework that will be used to develop the RESTful API. Flask meets all of the requirements necessary for the API aspect of the project. Flask-PyMongo is a library integrated with Flask that will also be useful when implementing communication between the API and MongoDB database so that database transactions can be performed.

SimPy

SimPy is a python framework used for process-based discrete-event simulations. This framework will allow the implementation of the back-end simulator that will generate simulation objects. The behaviour of malware can be modelled as a process which then can allow interaction with the SimPy environment. This allows environment timeouts to be triggered for a set time whenever a specific phase of the malware's release has been started. Alternatively, Cooja (contiki-os.org/start.html#start-cooja), OMNeT++ (omnetpp.org) or any other discrete-event simulator could potentially be used but SimPy has been chosen because of the fact that it has all of the requirements necessary to perform simulations, it is highly regarded and it is written in Python; which would mean that all of the back-end modules could be standardised to using Python.

Data Storage

MongoDB

MongoDB (mongodb.com) is the ideal candidate for the back-end database. The NoSQL solution will be able to handle potentially diverse data types contained in the data retrieved from Shodan and also the ability to scale the project if it ever became necessary to simulate on a large amount of devices. It also integrates with

PyMongo, enabling the ability of database transactions between the database and back-end server.

Dependency Management and Version Control

venv

venv (virtualenv.pypa.io) allows the creation of an isolated environment to handle python binaries and dependencies. Therefore, venv will be used for each Python-based module.

pip

pip (pypi.org/project/pip) provides an easy solution for installing python dependencies and therefore it will also be used for all python-based modules.

yarn

yarn (yarnpkg.com) offers a simplified way of managing projects and dependencies for JavaScript based applications and so it will be used when developing the user interface.

git

git (git-scm.com) is the de facto standard for version control. It will provide a way to manage the whole projects codebase in a more organised fashion.

Tooling

PyCharm

PyCharm (jetbrains.com/pycharm) is a Python-based Integrated Development Environment (IDE). Using PyCharm will allow the development of each Python-based module with speed and efficiency. PyCharm also provides various features such as debugging that will come in use during the development of this project.

Visual Studio Code

Visual Studio Code (code.visualstudio.com) will be used to develop the user interface part of the system. It is a highly extensible code editor with a multitude of features through community plugins as well as an interface for version control and a debugger.

ShareLaTeX

To write this document and any other additional documents ShareLaTeX (sharelatex.com/project) will be used, a LaTeX editor that bypasses the necessities of LaTeX installation and configuration.

Sketch

Sketch (sketchapp.com) was used to construct the UI designs [21]. Sketch is a design toolkit that is quickly increasing in popularity for designing user interfaces.

Resources

Shodan

Shodan (shodan.io) is a search engine for internet-connected devices, it is the site that will be used to retrieve the devices dataset. Shodan also offers an educational plan that will enable the collection of a necessary amount of data needed to conduct simulations.

Chapter 4

Implementation

4.1 The Database

As previously discussed, MongoDB was the technology decided upon to use for the back-end database. The database was only used to store the device data retrieved from Shodan and devices were only ever retrieved from the database on the initial load of the user interface using *db.devices.find()*.

4.2 Collecting Data

Dependencies

To suitably manage dependencies for the data collection module venv was used to create a virtual environment and then pip to install all of the necessary dependencies as seen below.

```
certifi==2018.1.18
chardet==3.0.4
click==6.7
click-plugins==1.0.3
colorama==0.3.9
idna==2.6
pymongo==3.6.1
requests==2.18.4
shodan==1.7.7
urllib3==1.22
XlsxWriter==1.0.2
```

The two main dependencies that are used within the data collection module are pymongo and shodan, pymongo is used to communicate with the MongoDB database whilst shodan is used to communicate with Shodan's API.

PyMongo

In this instance, PyMongo was only used to store device data. The following piece of code shows how the library performs such an action.

```
client = MongoClient(<INSERT_HOST>, <INSERT_PORT>)
db = client[<DATABASE_NAME]
db[<COLLECTION_NAME>].insert(<DEVICE_DATA>)
```

Shodan

To retrieve data from Shodan, their python library was used. The library offered a simplified means of querying for data. Shodan also offered a free student deal which enabled access to 200,000 query credits; this was more than enough to fulfil the needs of the project. The following piece of code shows how the library can be used to execute a query.

```
api = Shodan(<INSERT_API_KEY>)
results = api.search(<INSERT_SEARCH_QUERY>, page=<INSERT_PAGE_NUMBER>)
```

Shodan's API provided the necessities to retrieve the dataset necessary to perform simulations. For example, to find all Linux devices with telnet (port 23) open the following search query could be used:

```
port:"23" os:"Linux"
```

Processing and Storing Data

Once device data has been retrieved from Shodan it was necessary to first check how satisfiable the data was, this is done to avoid inserting data with unwanted characteristics such as sparsity. The satisfiability check involved checking that the device contained an operating system as well as a location. If the device passed the satisfiability check it can then be processed into a Device object (see appendix B). The operating system is checked because most malware have an operating system that they run on - whether its Windows, Linux or macOS. Therefore, it's useful to check that this field is not empty. The location field is checked because the latitude

and longitude aspect of the location field is necessary to placing the device on a map. It was also necessary to store the raw data retrieved from Shodan as during the implementation of the simulator it became necessary to perform a keyword search over the raw data, this will be talked about later on. See appendix C for an example of what a device retrieved from Shodan looks like before it is processed into the Device object.

4.3 Developing the Simulator

Dependencies

To suitably manage dependencies for the simulator module, venv was once again used to create a virtual environment and then pip to install all of the necessary dependencies as seen below.

```
click==6.7
Flask==0.12.2
Flask-Cors==3.0.3
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
pymongo==3.6.1
simpy==3.0.10
six==1.11.0
Werkzeug==0.14.1
```

The three main dependencies at use in the simulation module are Flask, Flask-Cors, PyMongo and SimPy.

API

Flask was used to implement the RESTful API that was necessary to integrate the simulator, database and user interface. Flask is simple and lightweight making it an appropriate choice to implement this aspect of the application. In addition to Flask, Flask-Cors was also used. Flask-Cors adds a Cross-Origin Resource Sharing (CORS) mechanism to the web server meaning that access to the API from a server on a different origin could be allowed. This would become necessary when making requests from the user interface. Additionally, PyMongo was used in the implementation of the

API. PyMongo is a library previously discussed that would allow the API to perform transactions on the back-end database. A simplified example of Flask and PyMongo working together can be seen in appendix A.

API Endpoints

The following endpoints are made available by the implemented RESTful API, the following section outlines each endpoint and gives examples of expected responses.

GET /malware/<name>

This endpoint returns a specific strain of malware dependent on the name parameter specified. The malware object returned in the response can be seen in appendix D.

GET /malware

This endpoint returns all of the strains of malware contained in a knowledge base of malware that can be simulated.

GET /devices/<id>

This endpoint returns a device stored in the back-end database with an id equal to the id parameter given. The device object returned in the response can be seen in appendix B.

GET /devices

This endpoint returns all devices stored in the back-end database.

POST /simulator

This endpoint returns a simulation object created from running a simulation using required parameters sent in the request body, the request body and the format of a simulation object that is returned can be seen below.

Request Body:

```
{
    malware: <MALWARE_NAME>,
    devices: <DEVICES>,
    params:{
```

```
            location: <LOCATION>
        }
    }
```

Format of Response:

```
    {
        "id": <ID>,
        "request": {
            "malware": <MALWARE>,
            "devices": <DEVICES>,
            "params": <PARAMS>
        },
        "events": <SIMULATION_EVENTS>,
        "statistics": {
            "total_devices": <TOTAL_NUMBER_OF_DEVICES>,
            "total_infected": <TOTAL_NUMBER_OF_INFECTED_DEVICES>,
            "total_clean": <TOTAL_NUMBER_OF_CLEAN_DEVICES>,
            "run_time": <RUN_TIME_OF_SIMULATION>
        }
    }
```

Simulator

As previously discussed, whenever a POST request is made to the RESTful API, a simulation using the SimPy framework is triggered according to the given parameters. To start a simulation a SimPy environment is first created, the set of devices being run on is then looped through and for each device a Markov chain transition matrix is generated and then a Markov chain (as seen in 3.15) is run using the transition matrix as input. The Markov chain returns an ordered list of all visited states, which in this case is the malware kill chain stages executed. These stages are then looped through and a SimPy environment process is created which adds a simulation event correlating to the current stage in the simulations events list, dependant on the running stage an environment timeout may be triggered to account for the time the stage has taken (i.e. a reconnaissance stage may take anywhere from 0.1 to 2 seconds). SimPy took time to understand due to it's use of generators, an unfamiliar concept. Generator functions allow the declaration of a function that behaves like an iterator, this means that it can then be iterated over in a loop.

Malware

In order to simulate the release of a strain of malware over a set of devices an encoded version of the released malware is necessary, this is then given to the simulator as a parameter. This allows the simulator to easily compare the malware's specification against the devices specification during the reconnaissance stage and also allows the simulator to attempt to gain access to the device using the attack vector that the malware tries to exploit. Appendix D shows how a malware object is encoded and appendix E shows an example of Mirai being encoded using the malware object.

Generating a Transition Matrix

When a transition matrix that follows the probabilities shown in the transition matrix table in 3.1 is generated during the simulation process the transition matrix needed to know how to generate the values that are dependant on data supplied (denoted *). The following points explain the way in which they were generated.

- $P(RS|R)$ - The probability of the reconnaissance stage being successful:
  This can either be 1 or 0 as a strain of malware can either run on a device or it cannot. This is usually down to properties such as the operating system but some malware target other characteristics. For example, Mirai runs on Linux but also targets specific types of devices such as Modems, Routers and IP Cameras. Therefore to determine the probability of malware being able to run on a device the specification of device that malware targets and the devices specification is compared, if every feature of the malware's target device specification is met by the device then the probability of achieving success at the reconnaissance stage is declared as 1. If not, it is declared as 0. Using the probability of success, the probability of failure can then be calculated using $1 - P(RS|R)$.

- $P(IS|I)$ - The probability of the intrusion stage being successful:
  Using the data provided and further research a probability of success can be predicted. As discussed in sections 2.3.4.1, 2.3.4.2 and 2.3.4.3 it was mentioned that Mirai, Brickerbot and BASHLITE brute-force the Telnet protocol using factory defaults to gain access. Knowing this information a target device can be analysed to see if it meets the requirements of the attack vector that the malware exploits (in this instance, is the device running Linux and is port 23

open?). If it does, the probability of success can be raised by 0.15. Using statistics gathered related to the attack vector the probability can then be raised even further. For example, on average 15% of IoT device owners do not change their factory defaults[20]. This statistic can allow us to up the probability of success by 0.15. Therefore, if a device meets the requirements of the Telnet attack vector then the probability of success is declared as 0.3. Using this calculated probability of success, a probability of failure can then be calculated using $1 - P(IS|I)$.

Running the Markov Chain

Once a transition matrix has been generated, the Markov chain can be run using the transition matrix as input. A state moves from one state to the next dependant on a random number generated. Once an absorbing state has been reached the chain will return an ordered list of visited states. See appendix F for the Markov chain code.

4.4 Building the User Interface

Building the user interface as a web application was an early decision made because of the fact that the project could then be freely accessible to anyone that wanted to know the vulnerabilities of IoT devices. However, if it ever became necessary to port the application to desktop that would be possible using Electron (`electronjs.org`). Electron allows the development of desktop applications using web technologies such as HTML and JavaScript through it's use of Chromium and Node.js.

Dependencies

To suitably manage dependencies for the user interface module yarn was used. All of the dependencies used for for the development of the user interface can be seen below.

```
"axios": "^0.18.0",
"leaflet": "^1.3.1",
"material-ui": "^0.20.0",
"prop-types": "^15.6.1",
"radium": "^0.24.0",
"react": "^16.3.2",
```

```
"react-dom": "^16.3.2",
"react-leaflet": "^1.9.1",
"react-leaflet-div-icon": "^1.0.2",
"react-redux": "^5.0.7",
"react-redux-loading-bar": "^4.0.3",
"react-scripts": "1.0.14",
"react-scrollbar": "^0.5.4",
"redux": "^3.7.2",
"redux-logger": "^3.0.6",
"redux-promise-middleware": "^5.0.0",
"redux-thunk": "^2.2.0"
```

Components

Using React the user interface was developed in a component-based fashion. This section details the major components in the user interfaces codebase that are critical to performing a simulation.

Map

The map component is used to display the Leaflet map with all of the given device markers on CartoDB's Dark Matter map shown in figure 3.6.

Menu

The menu component is used to display the side-menu shown in figure 3.10. When the criteria is given for a simulation using the drop-down lists, a simulation is then started by pressing the release button.

Device

The device component acts as a marker that can be displayed on the Map component. Each marker is placed at the location (latitude, longitude) specified within the given device data. A marker changes colour when hovered and when selected a device marker displays information about the selected device in a dialog box. The markers are represented as circles, which is different to the markers shown in figures such as 3.11. This is because leaflet's more efficient circle markers were used and they do not allow the use of images as markers. A device markers colour is also dependant on it's state in the running simulation. It's origin colour is white and then if a device gets

infected it is changed to red. If a device has suffered an unsuccessful infection then it's colour is changed to yellow.

Logger

The logger component acts as the main simulation aspect, it notifies the map whenever a device has been affected by a simulation so that the map is aware of how to colour each marker and it also logs all of the simulation events to the screen similarly to the logging shown in figure 3.12. The logger is essentially the simulation player, it is given a simulation object with all of the simulation events and it loops through these events executing them at the correct time each event occurs. Each event logged to the screen is made up of a timestamp, device endpoint (IP and port), location, device type, product name, operating system and an event message (i.e. "Device successfully infected with X").

Result

The result component is displayed after a simulation has finished and looks similar to the design shown in figure 3.13, it details the statistics of the simulation such as how many devices were simulated on, infected and clean as well as the run time of the simulation and it contains a "view logs" button that opens up the simulation logs in JSON format within another tab.

Redux

Redux was a key part of the user interface implementation. Redux works incredibly well with React because UI can be described as a "function of state" and then Redux can emit state updates in response to actions[18]. Redux was used to manage the retrieval of malware, devices and simulations through async requests made to the back-end RESTful simulator API. These requests were made using axios, a Promise-based HTTP client. Understanding the core concepts of Redux took a while to understand due to their complexity but after thorough reading of the Redux documentation (`redux.js.org`), the library was able to be used in a correct manner.

ACTION: Fetch Devices

When the application is first loaded, a "FETCH_DEVICES" action is dispatched to Redux and an async request using axios is made to the API. This allows Redux to tell the UI that devices are currently being fetched and because of this an indeterminate

loading bar is displayed. If the request is successful then a "FETCH_DEVICES_FULFILLED" action is dispatched to Redux, the loading bar is no longer displayed and the UI is able to display the devices as markers on the screen. If the request fails then a "FETCH_DEVICES_REJECTED" action is dispatched to Redux and the loading bar is no longer displayed.

ACTION: Fetch Malware

When the menu is first opened, a "FETCH_MALWARE" action is dispatched to Redux and an async request using axios is made to the API. If the request is successful then a "FETCH_MALWARE_FULFILLED" action is dispatched to Redux, the UI displays the malware options within the malware drop-down list in the menu. If the request fails then a "FETCH_MALWARE_REJECTED" action is dispatched to Redux.

ACTION: Fetch Simulation

When the release button in the menu is selected, a "FETCH_SIMULATION" action is dispatched to Redux and an async request using axios is made to the API. This allows Redux to tell the UI that a simulation is currently being fetched and because of this an indeterminate loading bar is displayed. If the request is successful then a "FETCH_SIMULATION_FULFILLED" action is dispatched to Redux, the loading bar is no longer displayed and the UI is able to start the simulation. If the request fails then a "FETCH_SIMULATION_REJECTED" action is dispatched to Redux and the loading bar is no longer displayed.

Chapter 5

Quality Assurance

This section outlines how assurances were made over the quality of the product developed in order to give enough confidence that it had reached a good standard.

5.1 Testing the User Interface

There are a multitude of ways the user interface could have been tested including the use of automated tests using libraries such as Jest (facebook.github.io/jest/en), however due to time constraints only manual testing was performed. This section outlines each test case and their results.

Summary	Steps	Expected Results	Actual Results	Status
Clicking the menu button opens up the menu	1. Click the menu button in the top left hand corner	The menu slides in from left to right	The menu slides in from left to right	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking a device marker opens up a dialog box displaying information about the device clicked	1. Click a circle device marker on the map	The device information dialog opens with information about the selected device	The device information dialog opens with information about the selected device	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking the release button starts a simulation according to the given criteria	1. Open up the menu 2. Fill in the simulation criteria 3. Click the release button	A simulation starts according to the specification given	A simulation starts according to the specification given	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking the information button on the menu opens up the information dialog	1. Open up the menu 2. Click the information button	The information dialog opens	The information dialog opens	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking the information button on the simulation logger opens up the information dialog	1. Open up the menu 2. Fill in the simulation criteria 3. Click the release button 4. Once a simulation has loaded, click the information button on the simulation logs section	The information dialog opens	The information dialog opens	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking the cancel button on the simulation logger cancels the current simulation	1. Open up the menu 2. Fill in the simulation criteria 3. Click the release button 4. Once a simulation has loaded, click the cancel button on the simulation logs section	A simulation is cancelled and the system is returned to it's original state	A simulation is cancelled and the system is returned to it's original state	Passed

Summary	Steps	Expected Results	Actual Results	Status
Dragging on the map with in a specific direction drags the map in the correct direction	1. Click and hold the map 2. Drag the mouse from left to right	The map moves in the left direction	The map moves in the left direction	Passed

Summary	Steps	Expected Results	Actual Results	Status
Once a simulation ends, the results screen displays to the screen	1. Click the menu button 2. Fill in the simulation criteria 3. Click the release button 4. Wait for a simulation to finish	The results screen displays to the screen	The results screen displays to the screen	Passed

Summary	Steps	Expected Results	Actual Results	Status
Clicking the view logs button on the results dialog opens up the simulation logs in a new tab	1. Click the menu button 2. Fill in the simulation criteria 3. Click the release button 4. Wait for a simulation to finish 5. Click the view logs button on the results screen shown	The simulations logs open in a new tab	The simulations logs open in a new tab	Passed

Summary	Steps	Expected Results	Actual Results	Status
Closing the result dialog reverts the system to it's original state	1. Click the menu button 2. Fill in the simulation criteria 3. Click the release button 4. Wait for a simulation to finish 5. Exit the result dialog by clicking outside of the dialog window	The system reverts to it's original state	The system reverts to it's original state	Passed

Summary	Steps	Expected Results	Actual Results	Status
Running a simulation on 5000 devices shows no performance issues	1. Ensure 5000 devices are stored in the backend database 2. Click the menu button 3. Fill in the simulation criteria 4. Click the release button 5. Wait for a simulation to finish	The simulation completes without showing any noticeable performance issues	The simulation completes without showing any noticeable performance issues	Passed

Chapter 6

Results and Evaluation

6.1 Fulfilment of Requirements

This section outlines all of the requirements and whether they were met in the final delivered product. For each incomplete requirement, unless otherwise stated, the requirement is incomplete because of a lack of time.

Essential

Summary	Type	Status	Notes
Display devices on a map	Functional	Complete	Devices are represented on the map as circle markers
Display specific device information	Functional	Complete	Clicking on a device marker opens up specific device information about the device clicked on
Simulate the effects of releasing a specified strain of malware	Functional	Complete	The user can select a strain of malware and simulate the release of the malware of a set of devices, feedback is given graphically by changing the device marker colours dependant on the state of the device as well as in a more detailed log history
Display the state of devices in a meaningful way	Functional	Complete	As previously mentioned, the state of a device is given by changing the colour of the devices marker with white being the default colour, yellow being the "failed to infect" colour and red being the "infected" colour.
A strain of malware should only be attempted to be released on a device that meets the malware's specification	Functional	Complete	The malware's target specification is compared with the device data and if the device meets all of the malware's specification it then the next stage of the kill chain is triggered

A strain of malware should only gain access to a device if the device has a highly probable chance of being vulnerable to one of the malware's exploits	Functional	Complete	Each strain of malware contains a set of exploits that it uses to try and gain access, these exploits are compared with the device data and a probability is determined
A relatively short description should be provided for each strain of malware	Functional	Complete	Whenever a strain of malware is selected in the menu, a link is displayed underneath the malware dropdown menu that when clicked, opens up a dialog displaying information about the selected strain
The user interface should have a high level of usability and accessibility	Non-functional	Complete	The user interface meets all of Nielsen's usability heuristics and follows Google's Material Design guidelines
The simulator should allow a large amount of devices to be simulated on with good performance	Non-functional	Complete	The simulator runs with no performance issues on 5000 devices

Table 6.2: Fulfilment of essential requirements

Desirable

Summary	Type	Status	Notes
A user should be able to specify a location in which to perform an attack	Functional	Complete	The user can either select "Worldwide" or a specific country to simulate an attack in
The user should be able to pause, play and stop a simulation	Functional	Incomplete	The user can stop a simulation but is unable to pause or play one
The user should be able to choose the speed of the simulation	Functional	Complete	The user can simulate at the following speeds; normal, 2x, 3x, 5x, 10x and 20x
The user should be able to add custom malware	Functional	Incomplete	A user can only simulate using the given options
The user should be able to add custom devices	Functional	Incomplete	A user can only simulate on the devices stored in the back-end database
The user should be able to switch between a light and dark interface	Functional	Incomplete	The user interface uses the dark interface but the option to switch between it and the light interface was not implemented

Each component should be run in an appropriate cloud computing service	Non-functional	Incomplete	During the development phase every component was ran locally and it was never necessary to run the components in appropriate cloud computing services due to the product not yet reaching the production stage

Table 6.4: Fulfilment of desirable requirements

6.2 Project Limitations

The main project limitation encountered was a "lack of data". The device data gathered from Shodan worked great when it came to calculating the probability of success at the reconnaissance stage of the kill chain detailed in 3.15. Further to this, after conducting research on malware, their attack vectors and gathering statistics a probability of success for the intrusion stage in the kill chain was able to be calculated sufficiently. However, it was impossible to predict success at the exploitation and action stages as it cannot be known if a target device has security measures in place such as anti-virus or intrusion detection systems (IDS) and if they do, how many? If data that gave information on devices and whether malware was able to infect them and carry out the malware's objective was available then a prediction on the probability of success using that data could have been calculated giving the simulation a more sufficient mathematical backing. Collating this data would have taken a very long time as access to a sandboxed IoT network and samples of malware would have been necessary. Due to this limitation, the probability of success at these stages was set to 1. Furthermore, data detailing the timings of each stage of the malware kill chain for the strains of malware used in the final implementation was not available. This meant that an estimation of these times needed to be given. Once again, collating this data would have taken a long time for the same reason.

6.3 Reflection on Management and Methodology

At the start of the project a decision was made to use Scrum as the project management methodology because the utilisation of Scrum's key principles would enable continuous improvement through constant reflection of progression. Working this way worked great and reading various Scrum for One resources by authors such as Alex Andrews from Ten Kettles (tenkettles.com)[2] and Dustin Wax, a previous

project manager at Lifehack (lifehack.com)[25] enhanced productivity and continuously improved results throughout the projects timespan. Furthermore, splitting each component up into separate modules as shown in figure 3.1 and using the dependency management and version control technologies discussed within 3.4 aided management of the projects codebase which served the project well.

6.4 Assessing the Implementation of Designs

User Interface

The initial user interface designs worked great when it came to implementing them, the colour scheme and the Dark Matter map supplied by CartoDB worked well together and integrated with Material Design elements it all connected beautifully. However, one aspect not taken into account when designing the user interface was the fact that device markers can layer on top of each other. This can affect the ability to click on certain device markers (if they are layered underneath another marker). To fix this a library was found that integrates with React-Leaflet called react-leaflet-marker-cluster (npmjs.com/package/react-leaflet-markercluster) which clusters nearby markers into one marker which when clicked expands to visibly show all clustered markers in a spiral fashion. Although this library seemed like it could fix this problem, using it then brought upon a new problem; how can the state of all devices be visibly shown during a simulation when some are hidden by marker clusters? As visualising the state of devices was deemed a higher priority than being able to click on every device marker for a detailed view of the selected device, a decision was made to not to use the marker cluster library (note: layered markers can still show their state due to the fact that the opacity of the device markers was set to 0.5). Furthermore, in the designs only the time, IP address and message of each simulation event are displayed (see figure 3.12). During the implementation phase this was changed to the layout below.

Time	Endpoint	Location	Type	Product	OS	Message

API

The API ensured a good means of communication between the user interface and database. However, implementing it as the middleman between the user interface and the simulator proved to be a bad decision. This is because simulation objects

generated by the simulator and sent back to the user interface can be incredibly large, and therefore sending them back over HTTP can be very inefficient. A more appropriate solution would be to setup a WebSocket connection between the client and server, this would allow the simulator to send frequent messages (or "events" in our case) to the user interface whenever a simulation is triggered. This would be much more appropriate and increase efficiency immensely.

Simulator

Designing a malware-specific kill chain and mapping it to the Markov chain that can be seen in figure 3.15 worked well when it came to the implementation. As discussed in section 6.2, the Markov chain was limited to only providing mathematically backed probabilities for the reconnaissance and intrusion stages. However, because of the way the Markov chain was implemented it will be easy enough to add in mathematically backed probabilities for the exploitation and action stages if it ever became a possibility.

6.5 Known Bugs

The user interface also contains a few bugs that due to time constraints unfortunately could not be fixed before submission, these are listed in the table below.

Summary	Severity
At end of simulation, when device markers are reset to their original state one marker is left at its previous state	Low
Occasionally the menu button and zoom in/out buttons disappear after a simulation	Medium
Occasionally during simulations, the simulation log does not automatically scroll to the bottom	Low
The zoom in/out button does not zoom in/out to/from the same coordinates but instead to coordinates north of the origin	Low

Table 6.5: Known bugs

6.6 Final Product

The final product fulfilled the main projects aim of demonstrating the vulnerabilities of the Internet of Things. However, if a user could create their own malware and add their own device dataset that contained devices that might be affiliated to them, the

product would be a lot more useful. Further to this, if the mathematical backing of the project was also improved the project may even have the potential to be used in industry. However, this was unfortunately not something that was able to be improved due to reasons discussed in section 6.2.

Figure 6.1: A screenshot of the final product in action

In hindsight, the way simulations finish should have been implemented differently. Currently, when a simulation ends the results screen is shown to the screen, giving statistics on the simulation and the ability to view the simulation logs. When this simulation results window is closed, every device marker is returned to it's default state rather than being kept at the state it ended the simulation in. This means that the user can not take time to analyse the "state of play" at the end of a simulation. This should be changed by not resetting the state of device markers at the end of a simulation and instead resetting them when a new simulation is started.

A problem that was identified when conducting simulations was that there was no indication of the progress of a simulation, this would be a useful feature as it would allow the user to estimate how long a simulation would have left.

Although only a few bugs were noticed (as discussed in section 6.5), confidence in the system is low because of a lack of testing conducted (see chapter 5). Ideally, more manual testing should have been administered as well as a good amount of automated testing such as unit tests and component tests. Additionally, it would have been ideal to also perform some integration testing.

6.7 Usage of Technologies

I believe the selection of technologies at use in this project were adequate, they provided the necessary means of executing the projects idea in an efficient manner. Usage of technologies such as MongoDB ensured scalability from a data storage point of view. Implementing the user interface as a web application using technologies such as React and Leaflet allowed simulations to be run on 5000 devices, this is largely down to the use of Leaflet's circle markers rather than its regular less efficient markers. Further testing would be necessary to measure how scalable the simulator actually is. Using the discrete-event simulation library SimPy worked well with generating simulation objects, however as previously mentioned in section 6.4; sending generated simulation objects back over HTTP using Flask did not turn out to be the best idea because of the size that simulation objects can be. Ideally, setting up a means of communication between the user interface and simulator over a WebSocket connection using SocketIO (`socket.io`) would have worked better. SocketIO has a JavaScript-based library and also a library that integrates with Flask, therefore the future implementation of this WebSocket connection would be fairly simple.

As previously discussed in section 6.6, confidence in the system is not as high as it should be due to a lack of testing, to carry out automated tests there are an abundance of libraries that could have been used. For the unit tests on the python-based modules pytest (`pytest.org`) could have been used. To perform component testing on the user interface, Jest (`facebook.github.io/jest`) could have been used.

6.8 Analysing the Vulnerability of the Internet of Things

Using the developed software, studies were conducted by running multiple simulations with three different strains of malware; Mirai (see section 2.3.4.1), Brickerbot (see section 2.3.4.2) and BASHLITE (see section 2.3.4.3). A decision was made to primarily focus on botnets because they have proven in recent years to be the most destructive to IoT (as is proven by the Dyn attack discussed in chapter 1). The results of these simulations can be seen below in tables 6.7, 6.9 and 6.11. Before each simulation occurred a suitable amount of unique devices were collected from Shodan that could potentially be vulnerable to the strain of malware being released. As all of the malware being used to conduct the studies necessary to meet the projects goal runs

on Linux and attempts to gain access using Telnet (port 23) the search parameters used to collect a dataset of devices (see section 4.2) were:

```
os:"Linux" port:"23"
```

Collecting data this way would mean that the reconnaissance stage of the Markov chain (see section 3.15) would have a high but not definite chance of being successful, this is due to the fact that these malware target specific types of devices as well.

Releasing Mirai into the Internet of Things			
Location	Total Number of Devices	Total Number of Infected Devices	Percentage of Infected Devices
Worldwide	4998	3331	66.64%
UK	144	130	90.02%
USA	2064	1630	78.48%
Canada	47	29	61.7%
Russia	204	116	56.86%
China	147	67	45.57%
France	39	19	48.71%
Germany	46	34	73.91%
Australia	7	3	42.85%
New Zealand	3	1	33.33%
India	46	12	26.01%

Table 6.7: Simulation results for Mirai

Releasing Brickerbot into the Internet of Things			
Location	Total No. of Devices	Total No. of Infected Devices	Percentage of Infected Devices
Worldwide	4998	3331	66.64%
UK	144	130	90.02%
USA	2064	1630	78.48%
Canada	47	29	61.7%
Russia	204	116	56.86%
China	147	67	45.57%
France	39	19	48.71%
Germany	46	34	73.91%
Australia	7	3	42.85%

New Zealand	3	1	33.33%
India	46	12	26.01%

Table 6.9: Simulation results for Brickerbot

Releasing BASHLITE into the Internet of Things			
Location	Total No. of Devices	Total No. of Infected Devices	Percentage of Infected Devices
Worldwide	4998	3331	66.64%
UK	144	130	90.02%
USA	2064	1630	78.48%
Canada	47	29	61.7%
Russia	204	116	56.86%
China	147	67	45.57%
France	39	19	48.71%
Germany	46	34	73.91%
Australia	7	3	42.85%
New Zealand	3	1	33.33%
India	46	12	26.01%

Table 6.11: Simulation results for BASHLITE

The results shown in tables 6.7, 6.9 and 6.11 the results are the same because all three strains of malware have the same requirements and use the same means of access. They were able to infect 3331 devices out of 4998 devices. With the Internet of Things making up almost eight billion devices[10], 4998 devices is an incredibly small sample of the network. Nonetheless, these results still prove that the IoT is vulnerable as there is a lot that can be done with 3331 insecure devices such as form a botnet and perform a DDoS attack on networks or critical infrastructure as is the purpose of Mirai and BASHLITE. Or alternatively, the devices could be bricked - as is Brickerbot's purpose. It should be noted that the these results do not show an accurate ratio of IoT devices in the given countries as that is not the purpose of the data collected from these simulations. The purpose of these results is to show that from a near 5000 devices collected, 66.64% of them were found to be probabilistically vulnerable. One last point worth making is that from the 2064 devices found in the United States, Mirai was able to gain access and exploit 78.48% of them.

Furthermore, the results produced that are shown in tables 6.7, 6.9 and 6.11 indicate how insecure the Internet of Things is which is the goal of this project.

6.9 Analysing the Accuracy of Simulation Results

Simulation's are powered by the data provided by Shodan and public information available on specific malware. Therefore simulations are only as good as the data available, for the most part the data from Shodan worked great and additional processing during the data collection phase ensured quality of data. Conducting manual analysis of simulation logs given from simulations improved confidence slightly in the results being given as these logs showed expected outcomes. For example, after simulating the release of Mirai against devices in the United Kingdom a sample of the logs generated that can be seen in appendix G shows the correct stages were executed. However, it is hard to determine why the intrusion stage (the event with the message, "Attempting to gain access to device") was successful, automated testing would help in this regard.

To predict the probability of the intrusion stage being successful for malware such as Mirai, Brickerbot and BASHLITE that bruteforce Telnet access using a list of default usernames and passwords (as discussed in section 2.3.4.1, 2.3.4.2 and 2.3.4.3) a statistic on the percentage of IoT device owners that do not change factory defaults was used. This allowed us to predict success with reasonable mathematical backing and in-turn, find 3331 vulnerable devices (see tables 6.7, 6.9 and 6.11). However, because SSH Communications Security recommends completely disowning Telnet and replacing the old protocol with the more secure and more recent, SSH. This is because a Telnet session between the client and the server is not encrypted and anyone with access to the TCP/IP packet flow between the communicating hosts can reconstruct the data that flows between the endpoints and read the messages, including the usernames and passwords that are used to log in to the remote machine[24]. This means that although the way successful access is predicted and calculated may not be adequate, devices can still be considered vulnerable if they are even running Telnet. Therefore, of the 4998 devices collected before gathering simulation results in 6.8 all 4998 can be considered vulnerable as they were all found to have had Telnet (port 23) open. The reason this is being mentioned is because the goal of this project was to demonstrate the vulnerabilities of the Internet of Things.

One of the main limitations as discussed in section 6.2 was a "lack of data" and this is the case again when comparing the simulation results in section 6.8 with

real results. If access to a dataset that identified devices that had been successfully infected by specific strains of malware was available then the data could be compared with simulation results to measure confidence.

Overall, confidence is lacking in the accuracy of current simulation results because of the reasons discussed but there are many ways confidence can be gained in the future as will be discussed in the next section.

Chapter 7

Future Work

7.1 Add Additonal Malware

Adding additional strains of malware to the malware selection criteria would allow users to conduct further studies using a range of diverse malware. These further studies could result in statistical findings answering questions such as "which operating system is the most vulnerable?".

7.2 Exception Handling

One aspect of the implementation that focus was not given to was exception handling, in the long run this could cause problems and so it is pivotal that back-end server exceptions that could possibly occur are handled appropriately and relayed back to the user interface with a human-readable message. What if the database is not up? How will the devices be retrieved? The user should be made aware of such an occurrence otherwise they will not be able to understand why the application is not functioning correctly.

7.3 Improving Markov Chain

Collecting more data that could assist with improving the Markov chain transition probabilities would improve the mathematical backing of the simulation and give the application more substance. In order to collate the appropriate data it would be necessary to test malware in sandboxed IoT networks. This would then mean data could be collected that could then be used in correlation with machine learning methodologies to predict the kill chain stage timings and transition probabilities for a given device.

7.4 Implementing Play/Pause Functionality

The implementation of play/pause functionality was a desirable requirement, the addition of this functionality would allow users to analyse the "state of play" at a specific point in time. This could aid in any analysis being conducted by the user or allow the user to pause a simulation and come back and play the simulation at a later point.

7.5 Implementing Simulation Progress

A feature that identifies the progress a simulation has made and estimates how long a simulation has left as discussed in section 6.6 would be extremely useful, especially for simulations on large amounts of devices because the user may not want to watch a whole simulation but instead know when to come back at the end of one.

7.6 Further Rigorous Testing

Further rigorous testing would improve confidence in the implementation and simulation results. As mentioned in section 6.9, automated testing could help to back-up simulation results statistically which would add an additional level of trust in the simulator.

7.7 Providing Possible Solutions to Vulnerabilities

Currently, the simulator just infects devices but does not provide information on why the device was infected. Adding reasons for infection and solutions to these given reasons could provide the user with a means of being able to counter vulnerabilities found.

7.8 Offering Personal User Profiles

User profiles would offer a way of personalising a users simulator, this section explains multiple ways this could be useful.

- Adding the ability to store previous simulations would allow the user to look back and run previous simulations.

- Implementing the addition of custom device insertion (and/or bulk device insertion) would allow the user to run simulations on a device set they have collated.
- Allowing the user to create custom malware would allow them to run simulations with strains of malware that may not already exist, which could in turn provide useful information to improve security before a similar strain of malware is let loose in the "wild".
- Adding alternative colour schemes such as the light scheme shown in figure 3.3 would allow the user to choose a colour scheme they find most usable and aesthetically pleasing.

Chapter 8

Conclusions

At the start of this project the main goal was to demonstrate the vulnerabilities of the Internet of Things. After examining how this could be done in an original and useful manner, a quote was found as mentioned in chapter 1. The quote explains that the fight against malware is missing a key ingredient; the implementation of software that simulates malware propagation. This led to the development of software that could simulate the propagation of malware on real internet-connected device data provided by Shodan (shodan.io).

Overall, the software developed succeeded in supplying the means to satisfy the projects main goal. The final product can be used to simulate a strain of malware being released on a set of devices and produce mathematically backed results. This enabled the ability to simulate on real IoT device data as shown in section 6.8, these simulations produced alarming results that definitely demonstrate how vulnerable the IoT is. However, the accuracy of these results can be questioned due to multiple reasons; these reasons are discussed in section 6.9. A part of the final product that was particularly pleasing was how much the user interface designs resembled the original designs; this meant that the initial designs were well designed to fulfil the projects requirements.

The project also resulted in useful work being done towards creating a piece of software that can simulate malware propagation such as the the malware kill chain model (inspired by the cyber kill chain as shown in figure 2.1) and the encoding of the model in a Markov chain that allows the addition of probability to the simulation. This model can then be used in correlation with device data to determine success. The user interface developed also provides an exceptional means of conducting simulations whilst also providing an enriching simulation experience. However as mentioned in chapter 6, confidence is still lacking in regard to the implementation due to a lack of testing and also a lack of data to provide accurate mathematically backed results.

Furthermore, working on this project has provided a sense of belief that the implementation of a malware propagation simulator that provides accurate results is possible and it is hoped that this study will stimulate further investigations in this field.

Chapter 9

Reflections on Learning

Using Scrum as the project methodology for this project allowed continuous improvement through constant reflection of progression. This enabled the use of "double-loop learning", a term coined by Chris Argyris. By retrospectively adapting assumptions, concepts and ideas dependant on realisations made from reflecting the project was able to repeatedly show signs of improvement. An example of this would be that when the implementation of the simulator was being conducted no discrete-event simulation library was being used initially, this proved to be highly unproductive and so the implementation was paused, further research was conducted on discrete-event simulation libraries and SimPy (`simpy.readthedocs.io`) was found. This increased productivity and enabled me to come more familiar with simulation best practices.

Throughout the large majority of the time spent on this project, the project was viewed as more of a software/web development-based project rather than a research project. In hindsight, more focus should have been aimed towards providing a more substantial mathematical backing rather than developing a fully fledged piece of graphical web-based software as well. It would have been more than adequate to completely scrap the user interface and develop the simulator as a command-line application; this would have allowed me to spend the time necessary to collate data that would have enhanced the accuracy of results.

After reflecting on the final product (see section 6.6) and results produced from performing simulations (see sections 6.8 and 6.9) an abundance of features and further ways in which the mathematical backing of simulations can be improved were discovered (see chapter 7).

Once results had been produced from simulations the next step was deciding how to represent these results in a way that could demonstrate best how vulnerable the Internet of Things is. This required some "trial and error" and reflection of different

solutions, eventually the result format shown in tables 6.7, 6.9 and 6.11 was decided on.

Beforehand, only a basic understanding of the Internet of Things, malware and how malware propagates through networks and systems was known. This project has improved knowledge and understanding of these fields and provided a greater passion for cyber security as a whole. Furthermore, this project has also given the author the experience of writing an academic paper - an experience found thoroughly enriching and rewarding.

Appendix A

```
from flask import Flask
from flask_cors import CORS
from pymongo import MongoClient

app = Flask(__name__)
CORS(app)
client = MongoClient(<INSERT_HOST>, <INSERT_PORT>)
db = client[<INSERT_DATABASE_NAME>]

@app.route('/')
def index():
    collection = db[<COLLECTION_NAME>]
    documents = collection.find()
    return documents
```

Appendix B

```
import json

class Device:
    def __init__(self, raw_result):
        self.ip = None
        self.location = None
        self.os = None
        self.port = None
        self.type = None
        self.product = None
        self.raw_result = raw_result
        self.process()

    def process(self):
        if 'ip_str' in self.raw_result and self.raw_result['ip_str']:
            self.ip = self.raw_result['ip_str']
        if 'location' in self.raw_result and self.raw_result['location']:
            self.location = self.raw_result['location']
        if 'os' in self.raw_result and self.raw_result['os']:
            self.os = self.raw_result['os']
        if 'devicetype' in self.raw_result and self.raw_result['devicetype']:
            self.type = self.raw_result['devicetype']
        if 'product' in self.raw_result and self.raw_result['product']:
            self.product = self.raw_result['product']
        if 'port' in self.raw_result and self.raw_result['port']:
            self.port = self.raw_result['port']
```

```
    return self

# Checks whether the device data reaches satisfiable standard to simulate o
def is_satisfiable(self):
    if self.os:
        if self.location:
            if self.location['longitude'] and self.location['latitude']:
                return True

    return False

def to_json(self):
    return json.loads(self.to_json_str())

def to_json_str(self):
    return json.dumps(self, default=lambda o: o.__dict__, sort_keys=True, i
```

Appendix C


```
{
    '_shodan': {
        'options': {},
        'id': None,
        'module': 'telnet',
        'crawler': '5faf2928ceb560cb4276cc1b4660b2d763cc6397'
    },
    'hash': 0,
    'os': 'Linux 2.6.x',
    'ip': 1137827181,
    'isp': 'Plateau Telecommunications Incorporated',
    'port': 23,
    'hostnames': ['067-209-221-109.plateautel.net'],
    'link': 'Ethernet or modem',
    'location': {
        'city': 'Clayton',
        'region_code': 'NM',
        'area_code': 575,
        'longitude': -103.355,
        'country_code3': 'USA',
        'country_name': 'United States',
        'postal_code': '88415',
        'dma_code': 634,
        'country_code': 'US',
        'latitude': 36.4016
    },
    'timestamp': '2018-05-10T05:21:29.287640',
```

```
    'domains': ['plateautel.net'],
    'org': 'Plateau Telecommunications Incorporated',
    'data': '',
    'asn': 'AS21782',
    'transport': 'tcp',
    'ip_str': '67.209.221.109'
}
```

Appendix D

```
class Malware:
    def __init__(self, name, specification, attack_vectors, actions, info):
        self.name = name
        self.specification = specification
        self.attack_vectors = attack_vectors
        self.actions = actions
        self.info = info
```

Appendix E

```
Malware(
    "Mirai",
    {
        "os": "Linux",
        "keywords": [
            [
                "Router",
                "Modem",
                "Camera",
                "IP Camera",
                "IPCamera",
                "Netcam",
                "linksys",
                "netgear",
                "Server: SQ-WEBCAM",
                "linux upnp avtech",
                "webcamxp"
            ]
        ]
    },
    [
        {
            'id': 'telnet',
            'type': 'bruteforce',
            'requirements': {
                'os': 'Linux',
                'port': 23
```

```
            },
            'probability_of_success': 0.15,
            'reason': 'Open Telnet port using default
            username/password combination'
        }
    ],
    "Device under control by command and control server",
    "Mirai was the strain of malware used to perform the
    famous Dyn attack. It works by scanning the internet
    for hosts with an open telnet port (TCP port 23).
    Once a host is identified it uses a list of default
    usernames and passwords to gain access to devices
    that are running BusyBox. Once access has been gained,
    the malware is installed and a Command and Control (C2)
    server is made aware. It then awaits further instructions.
    The C2 server can then be used to launch a flood of
    various kinds of traffic, overwhelming the a target host.
    Mirai mostly used infected CCTV camera devices to carry
    it out the Dyn attack. During it's scan for eligible
    hosts, Mirai also checks a 'do not mess with' list of
    servers that include General Electric, Hewlett Packard,
    and the U.S. Department of Defense. If the host is on
    the list, it is not targeted."
)
```

Appendix F

```
import random

class MarkovChain(object):
    def __init__(self, origin, transition_matrix):
        self.visited_states = [origin]
        self.current_state = origin
        self.transition_matrix = transition_matrix

        # Seeding helps to reproduce results
        random.seed(42)

    def run(self):
        running = True

        while running:
            if self.at_absorbing_state():
                running = False
            else:
                self.move()

    def move(self):
        legal_transitions = self.transition_matrix[self.current_state]
        r = random.uniform(0, 1)

        next_state = None

        cumulative_probability = 0.0
```

```
        for transition in legal_transitions:
            transition_probability = legal_transitions[transition]
            cumulative_probability += transition_probability
            if r < cumulative_probability:
                next_state = transition
                break

        self.current_state = next_state
        self.visited_states.append(self.current_state)

    def at_absorbing_state(self):
        available_transitions = self.transition_matrix[self.current_state]
        if self.current_state in available_transitions:
            if available_transitions[self.current_state] == 1:
                return True
```

Appendix G


```
{
  "time": "0:36:59",
  "endpoint": "88.211.68.169:23",
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Performing reconnaissance on device"
},
{
  "time": "0:37:00",
  "endpoint": "88.211.68.169:23",
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
```

```
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Device meets specification set by malware"
},
{
  "time": "0:37:00",
  "endpoint": "88.211.68.169:23",
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Attempting to gain access to device"
},
{
  "time": "0:37:05",
  "endpoint": "88.211.68.169:23",
```

```
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Access successfully gained to device"
},
{
  "time": "0:37:05",
  "endpoint": "88.211.68.169:23",
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Attempting to infect device"
```

```
},
{
  "time": "0:37:11",
  "endpoint": "88.211.68.169:23",
  "location": {
    "city": "Dulwich",
    "region_code": "M8",
    "area_code": null,
    "longitude": -0.08330000000000837,
    "country_code3": "GBR",
    "country_name": "United Kingdom",
    "postal_code": "SE21",
    "dma_code": null,
    "country_code": "GB",
    "latitude": 51.44999999999999
  },
  "type": null,
  "product": null,
  "os": "Linux 3.x",
  "message": "Device successfully infected with Mirai"
}
```

References

[1] Adam Entous, Ellen Nakashima and Miller, Greg. *Secret CIA assessment says Russia was trying to help Trump win White House*. 2016. URL: https://www.washingtonpost.com/world/national-security/obama-orders-review-of-russian-hacking-during-presidential-campaign/2016/12/09/31d6b300-be2a-11e6-94ac-3d324840106c_story.html?noredirect=on&utm_term=.b37813786a0b. (Accessed on 2018-05-07.

[2] Andrews, Alex. *Scrum Of One: How to Bring Scrum into your One-Person Operation*. 2017. URL: https://www.raywenderlich.com/162654/scrum-one-bring-scrum-one-person-operation. (Accessed on 2018-05-07.

[3] BBC. *Cyber-attack: US and UK blame North Korea for WannaCry*. 2017. URL: http://www.bbc.co.uk/news/world-us-canada-42407488. (Accessed on 2018-05-07.

[4] BBC. *The Interview: A guide to the cyber attack on Hollywood*. 2014. URL: http://www.bbc.co.uk/news/entertainment-arts-30512032. (Accessed on 2018-05-07.

[5] BBC. *Timeline: How Stuxnet attacked a nuclear plant*. URL: http://www.bbc.co.uk/timelines/zc6fbk7. (Accessed on 2018-05-07.

[6] CartoDB. *Introducing Positron and Dark Matter Basemaps from CartoDB*. 2014. URL: https://carto.com/blog/getting-to-know-positron-and-dark-matter/. (Accessed on 2018-02-10).

[7] Chirgwin, Richard. *Equifax reveals full horror of that monstrous cyber-heist of its servers*. 2018. URL: https://www.theregister.co.uk/2018/05/08/equifax_breach_may_2018/. (Accessed on 2018-05-07.

[8] CNN. *2016 Presidential Campaign Hacking Fast Facts*. 2018. URL: https://edition.cnn.com/2016/12/26/us/2016-presidential-campaign-hacking-fast-facts/index.html. (Accessed on 2018-05-07.

[9] Dictionaries, Oxford. *Definition of Markov Chain in US English*. URL: https://en.oxforddictionaries.com/definition/us/markov_chain. (Accessed on 2018-05-03).

[10] Gartner. *Gartner Says 8.4 Billion Connected 'Things' Will Be in Use in 2017, Up 31 Percent From 2016*. 2017. URL: https://www.gartner.com/newsroom/id/3598917. (Accessed on 2018-01-29).

[11] Goodin, Dan. *PlayStation Network breach will cost Sony $171m*. 2011. URL: https://www.theregister.co.uk/2011/05/24/sony_playstation_breach_costs/. (Accessed on 2018-05-07.

[12] Google. *Material Design*. 2014. URL: https://material.io/guidelines/material-design/introduction.html. (Accessed on 2018-02-09).

[13] Hospelhorn, Sarah. *The Cyber Kill Chain or: how I learned to stop worrying and love data breaches*. 2016. URL: https://blog.varonis.com/the-cyber-kill-chain-or-how-i-learned-to-stop-worrying-and-love-data-breaches/. (Accessed on 2018-05-07.

[14] Microsoft. *Defining Malware: FAQ*. 2003. URL: https://technet.microsoft.com/en-us/library/dd632948.aspx. (Accessed on 2018-05-07.

[15] Microsoft. *IoT for Healthcare*. URL: https://www.microsoft.com/en-gb/internet-of-things/healthcare. (Accessed on 2018-05-07.

[16] Nielsen, Jakob. *10 Usability Heuristics for User Interface Design*. 1995. URL: https://www.nngroup.com/articles/ten-usability-heuristics/. (Accessed on 2018-05-09.

[17] Perry, J Steven. *Anatomy of an IoT malware attack*. 2017. URL: https://www.ibm.com/developerworks/library/iot-anatomy-iot-malware-attack/. (Accessed on 2018-05-07.

[18] Redux. *Usage with React*. URL: https://redux.js.org/basics/usage-with-react. (Accessed on 2018-05-04).

[19] Rey, Angel MartÃŋn del. *Mathematical modeling of the propagation of malware: a review*. 2015. URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.1186. (Accessed on 2018-05-05.

[20] Shah, Sooraj. *15 per cent of IoT devices owners don't change the default password*. 2017. URL: https://www.theinquirer.net/inquirer/news/3012365/15-per-cent-of-iot-devices-owners-dont-change-the-default-password. (Accessed on 2018-05-09.

[21] Sketch. *A toolkit for designing user interfaces*. 2008. URL: https://www.sketchapp.com/. (Accessed on 2018-02-09).

[22] Solon, Olivia and Hern, Alex. *'Petya' ransomware attack: what is it and how can it be stopped?* 2017. URL: https://www.theguardian.com/technology/2017/jun/27/petya-ransomware-cyber-attack-who-what-why-how. (Accessed on 2018-05-07.

[23] Spring, Tom. *Bashlite Family of Malware Infects 1 Million IoT Devices*. 2016. URL: https://threatpost.com/bashlite-family-of-malware-infects-1-million-iot-devices/120230/. (Accessed on 2018-05-09.

[24] SSH. *Telnet - and SSH as a Secure Alternative*. 2018. URL: https://www.ssh.com/ssh/telnet. (Accessed on 2018-05-09.

[25] Wax, Dustin. *Scrum for One*. URL: https://www.lifehack.org/articles/featured/scrum-for-one.html. (Accessed on 2018-05-07.

[26] Woolf, N. *DDoS attack that disrupted internet was largest of its kind in history, experts say*. 2016. URL: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet. (Accessed on 2018-02-02).