4 Introduction

4.1 IoT technology adaptation

Modern society has seamlessly adapted to using gadgets and appliances that 30 years ago only existed in futuristic science fiction movies. Smart devices have slowly become a part of daily lives. Self-driving trucks are no more something out of a distant future[18] and the Internet connected appliances such as light bulbs and smart coffee makers are already finding their place in common peoples homes. It was estimated that in 2017 there were approximately 27 billion connected IoT devices around the globe. This number is expected to increase by approximately 12 % each year until it reaches 125 billion by 2030 [20]. For comparison, in 2014 there were around 1.57 billion smart phone owners worldwide, this number is bound to reach 2.87 billion by 2020 [3]. Despite the increasing number of smart phones, they are limited by the number of people using them. This is not the case with smart Internet of Things devices. For example, an industrial factory may contain hundreds of different sensors ranging from management department AC connected thermometers to ensure staff comfort and electricity consumption meters to pressurized explosive gas cylinder sensors that are vital to ensure safety and efficiency of the factory. The IoT networks and stand-alone "Smart things" technologies are booming [17] and will only increase in numbers. Unsurprisingly, such growth leads to privacy, security and regulation concerns as it is hard to control. Although these technologies have greatly complimented peoples lives, one cannot help wondering if their owners can manage these numbers of Internet connected devices properly.

4.2 Security flaws

Even though IoT devices collect incredible amount of users sensitive information and it is starting to cause processing and security concerns [29] little is being done to ensure proper security of these devices. As it has been shown by the Mirai botnet attack in 2016, IoT technologies can be used to create record breaking botnets and disrupt more traditional Internet services[13]. If proper measures are not taken it is likely that larger scale IoT network exploitation and malicious activities will take place. As manufacturers are being pushed by the market growth to develop new products faster to keep up with their competitors, proper security testing is becoming a sort of luxury that usually not everyone can afford. Moreover, it appears that consumers are not concerned about their device security[30]. As studies have shown even Internet connected vehicles can be hacked into and controlled remotely[23]. As Internet connected devices are exposed to many more threats than stand-alone electronic devices, steps must be taken to ensure that IoT gadgets and Internet connected devices are resilient and safe to use.

Due to vast differences in deployment environment as well as technologies in use, proper IoT testing is hard to ensure. One reason that severely complicates the testing process is the tendency of manufacturers to use third-party technologies and services during the development stages. A single IoT device firmware may be written by a mix of contracted developers known as Original Design Manufacturers (ODM) and in-house developers hired by hardware manufacturer - Original Equipment Manufacturer (OEM), then the application code itself may be supplied by a completely different company[19]. The problem occurs when OEM and ODM reaches the stage where their code bases must be merged. The ODM may provide only the binary files or an SDK for the OEM. Therefore, if that happens Original Equipment Manufacturer (OEM) which is responsible for distributing firmware, managing it and releasing updates, does not have full access to the code. Moreover, IoT networks comprise a big number of various devices that are responsible for diverse functions. Each device individually may be manufactured by a different supplier and their OEMs accordingly. As no individual link of the supply chain possesses access to the full infrastructure source codes, it may be impossible to thoroughly test the system as a whole. It becomes apparent that the black box type penetration testing and vulnerability scans may be the only acceptable strategy aiming to secure complicated IoT systems.

As the field in question is rather new, diverse and rapidly developing, it is hard to find knowledgeable specialists and tools developed specifically for IoT penetration testing. There are numerous blogs and projects offering a various level of detailed guides to IoT penetration testing[33]. Some organizations including IEEE and ETSI have released technology-specific standards as well as security guidelines but none of them have tried to cover IoT in general[35].

4.3 Project goals

The purpose of this project is to simplify IoT penetration testing and provide a framework for future development. The project summarizes general IoT cybersecurity assessment concepts and presents them in a user-friendly manner. Moreover, this project aims to address the lack of dedicated IoT penetration testing software and provides an extensive framework prototype whose purpose is to reuse the existing penetration tools to test IoT systems. The tool is designed for users from software development background but with little knowledge of cybersecurity or pen testing.

As the IoT technology stack is so diverse it is impossible to design a tool that covers all the cases of its use. Instead, this project aims to provide a "tool box" of well-known and proven penetration tools and has the function to conveniently add new tools in case of need. The application is aimed to be highly customizable and coherent.

Finally, in order to simplify the re-use of the proposed threat model and offer a practical solution, the developed penetration testing tool prototype must be closely bound with the IoT threat model concepts. A way of conveniently linking the information stored in the threat model and the "tool box" tools would greatly increase the prototype usability and justify its development.

5 Background research

5.1 Penetration testing

Penetration testing in general is a controlled form of hacking in which the tester acts as an attacker in order to find system misconfigurations and bugs that may lead to security threats[7]. There are generally two approaches to penetration testing: White and Black box. White box testing is called structural testing because the test cases are usually executed by software developers that are aware of internal code structure. Black box testing corresponds to functional testing and is intended to be performed without prior knowledge of the software internal mechanisms. The Black box pen testers focus on testing application functionality and are only concerned about program input and output[25]. It has been noted that by combining these two methods it is possible to systematically discover target vulnerabilities and propose remediation[32]. It has further been proposed that penetration testing must not be regarded as the final stage before releases but become an integral part of the development cycle in order to prevent similar vulnerabilities from appearing in future applications[15].

There are numerous books written about penetration testing[34], best practices regarding web applications, embedded systems and networking but only a limited amount of information is available about IoT penetration testing specifically[19]. IoT technologies just combine and modify previously developed solutions rather than inventing completely new technologies. The shift in use cases of well-developed mechanisms and the amount of possible variations in the IoT environment requires a distinct look at its security. Therefore, there is a need to adapt parts of classical infrastructure pen testing to a rather distinct IoT field. Another important reason is to make penetration testing less complicated for developers and people with less in-depth security related knowledge as well as create a convenience tool for existing penetration testers.

5.2 Internet of Things

There is no clear definition for the phrase Internet of Things, it is reasonable to define it as "the concept of every device blending with the existence of human beings"[24]. This means that smart devices mimic human interaction and other systems would not be able to distinguish human interacting with it from another system. In its simplest form IoT system definition can be rephrased as a decentralized network where multiple, usually, limited capabilities embedded processing units communicate between each other in various ways[31]. The fact that they have communication capabilities implies that they can change their behaviour depending on their network interface input and can essentially be controlled remotely. That is what exposes "smart objects" to outside threats[14]. Therefore, embedded system engineers, that were used to developing independent devices or small, single purpose closely bound networks, nowadays need to consider what consequences their decisions may have on the overall user infrastructure. Similarly, the penetration testing complexity increases relatively to the system size and technology set used. The addition of a network interface converts a narrow purpose device into an interactive Internet component.

Inserting a previously isolated technology into the global Internet network attracts the attention of users with malicious intentions. Unprotected log-ins, outdated software and insecure communication would not cause major security issues for hidden away, stand-alone embedded devices if they function properly (e.g. medical equipment). The situation changes radically if a device can be discovered and interacted with by anyone on the network. IoT devices can be desirable and favourable targets for hackers that are expanding their bot nets. Unlike regular computers, IoT devices usually run continuously round-the-clock and can be exploited without the owners knowledge[26]. It seems that traditional approaches of securing devices are not effective[28] due to IoT technology uniqueness and variety. Therefore, penetration testing is proposed to imitate hacker attacks and evaluate IoT technology security near to real world environment.

5.3 IoT penetration testing

Above listed IoT inherited vulnerabilities require pen testing to be performed for every system layer, breaking down IoT network infrastructure and exposing hidden attack vectors. Testers must consider five different aspects of IoT infrastructure. Hardware vulnerabilities can be exploited by anyone who has physical access to the device. Such vulnerabilities may be an open debugging port, password reset button and tapping into hardware level communication (e.g. UART)[2]. Firmware in this context stands for device operating system which can be rather primitive or extensive and may be using third party SDK and libraries that could introduce possible vulnerabilities[19]. Application level threats usually happen due to a software bug or a logic error[19]. If a device or an IoT system has a web interface, it essentially becomes a web host; thus, it may have all the vulnerabilities of a regular website[1]. Communication and network exploits occur due to man-in-the-middle attacks and usage of insecure communication protocols assuming security by obscurity. Lastly, some IoT vendors release mobile applications complimenting their products. That is another new and troublesome attack vector as hackers may get access not only to the victims IoT devices but also get a foot hold in users smart phone and might be able to access personal information stored in there[19]. Only by addressing each part of the infrastructure individually and later as a whole one can thoroughly test an IoT system.

5.4 Threat modelling

Threat modelling is a technique to identify the weak spots of the infrastructure and suggest countermeasures for any vulnerabilities discovered in the process. When performed and maintained from early stages of development it helps to map the likely system vulnerabilities that may result in a malicious event that may compromise the systems integrity[21]. For some small- scale applications threat modelling might seem inefficient and unnecessary as the system assets are not numerous and they do not have any complicated relations. For larger scale enterprise applications threat modelling has become a crucial part of development process and a valuable asset for risk management continuum[22].

Over time many different threat modelling approaches have been developed in order to adapt to the distinct enterprise fields where they were used. The author of the book "IoT Penetration Testing Cookbook"[19] suggests using a well-known STRIDE threat model in combination with the DREAD threat evaluation model to describe the likely IoT system vulnerabilities. The author of the book also suggests simply listing each individual asset vulnerabilities (later explained in more detail) as an alternative to rather binding STRIDE framework[19].

5.4.1 STRIDE

The STRIDE threat modelling framework has been developed by Microsoft and divides security threats to six categories[12] in accordance to each letter in the name.

- Spoofing of user identity
- Tampering - altering equipment in order to cause malicious behaviour
- Repudiation - ability to modify information without taking responsibility or being detected
- Information disclosure - improper management of sensitive information
- Denial of Service - degrading the quality or eliminating the service
- Elevation of Privilege - gaining system rights without proper authorization

A Data Flow diagram is expected to be used together with STRIDE model. This way every node of the system under consideration can be visualized and examined separately. After thorough analysis it is expected to have a full list of system vulnerabilities.

5.4.2 STRIDE alternative

Plainly listing node vulnerabilities of every system may seem unorderly and counter intuitive but there are cases where such solution is advantageous. It follows Agile development principles relatively closely and is, although not directly referenced, similar to Abuser Stories threat modelling method. As the name might suggest the tester is encouraged to think as an attacker and design attacks that could later on be launched on the system in hopes of confirming the raised hypothesis for the existence of specific vulnerability. Abuser stories method emphasizes finding the possible system entry points that could be used by the malicious intent attacker. It also requires that for every suspected threat a possible mitigation technique and its accompanying test must be specified[27].

A dynamic approach like this may be rather appealing in IoT environment where threats sometimes cannot be clearly assigned to one particular group. It may also be complicated to distinguish system components due to the vast variety of the possible IoT system elements. As each individual device is usually small and of limited capabilities the overall system model and relations between its nodes may provide much more information than in-depth analysis of each node.

5.4.3 DREAD ranking

DREAD ranking is a risk assessment model introduced by Microsoft and now used by OpenStack and advertised by OWASP[10][5]. The name DREAD stands for five categories which are used to estimate every threat.

1. Damage potential - How great is the damage if exploited?
2. Reproducibility - How easy is it to reproduce the attack?
3. Exploitability - How easy is it to attack?
4. Affected users - How many users roughly are affected?
5. Discoverability - How easy is it to find the vulnerability? [19]

For every category a rating from 0 to 10 is derived, when all the ratings are summed up and divided by five; a decimal score is produced where the highest scores represent the most urgent vulnerabilities.

A variation of this model is suggested by some sources[16][19] where the threats are ranked on the scale from 0 to 3 and the impact is evaluated from low to high:

Risk rating	Result
High	12 - 15
Medium	8 - 11
Low	5 - 7

Table 1: DREAD ranking

5.5 IoT threat modelling

"IoT penetration testing cookbook"[19] suggests, to perform IoT threat modelling in these steps:

5.5.1 Identifying system assets

In this context an 'asset' stands for any physical or virtual device which is responsible for a certain part of the system. For each device, document all publicly available information that may provide any hints on the internal processes of the system.
In a home security system setup good examples of 'assets' would be surveillance cameras, a digital video recorder, web application used to access DVR's content, cameras and DVRs firmware.

5.5.2 Architectural overview

IoT architectural overview goal is to describe system functionality, use cases and physical composition. It helps to visualize how an intruder may use the system resources in an unintended manner. The point is to discover flaws in system design or implementation. This can be achieved in three steps:

1. Documenting system functionality and features

   This can be achieved by listing several use cases for different system stakeholders.

2. Creating an architectural diagram of system infrastructure

   There are plenty of UML diagram building tools available online that could be used for this purpose (e.g. www.draw.io www.lucidchart.com).

3. List all the technologies used in the system

   In this section a 'technology' can be anything that was developed for a particular purpose and may have inherited vulnerabilities.
   Good technology examples are a Wi-Fi router, an android or iOS mobile application, an embedded Linux OS, various communication protocols.

5.5.3 IoT device decomposition

This is useful in order to map system interaction points and document data transition between system components. This step corresponds to mapping attack surfaces in penetration tester terminology[9].

1. Creation of data flow diagram

   A data flow diagram can be an extension of architectural diagram discussed previously just with more details concerning information movement and system trust boundaries.

2. List of system entry points

   The entry points are listed in a similar manner as technologies or assets but with emphasis on the system and user interaction as well as data transition between infrastructure nodes.
   System entry point examples are embedded web applications, mobile applications, physical connections to the DVR, cameras (as they are registered and communicate with the DVR)

5.5.4 List of potential threats

At this stage, the pen tester is supposed to use the previously gathered information to list all the potential threats for each of the system components. In order to thoroughly analyze the system model, it is helpful to do it two fold: as the vendor concerned about user data security and application integrity and as the attacker searching for exploitable targets to compromise network integrity.

STRIDE can be used to thoroughly consider node vulnerabilities of every system, it can also help to organize threats when the threat model becomes bigger. As an alternative, a dynamic approach discussed above, is also acceptable.

Each threat is described in four steps:

1. Threat description
2. Threat/attack target
3. Possible attack techniques
4. Suggested countermeasures

At this stage the information entered does not have to be exact or even entirely correct, it is more important to have an exhaustive list of potential vulnerabilities that can be updated later.

5.5.5 Threat ranking

In order to prioritize vulnerabilities, the previously explained DREAD method is suggested. There are other popular ranking systems like CVSS[4] that can be used. After sorting the threats based on their risk rating, it is easier to prioritize high-risk potential vulnerabilities and focus on their testing.

5.5.6 Mitigation, testing and re-evaluation

The penetration testers are encouraged to maintain system risk model up-to-date as it provides a convenient framework for documenting testing progress. New information can help to discover related threats or uncover unexpected entity relations.
If possible, providing automated vulnerability tests is beneficial, although it is not always possible. In order to maintain secure IoT infrastructure it is highly recommendable to re-iterate thorough threat modelling process after every new release or a functionality update[15].

6 Project specification and analysis

This section follows the problem analysis process and defines the project scope listing its requirements. Alternative problem approaches are then discussed, followed by justification of the chosen solution.

6.1 Alternative solutions

- **Paper based tables and notes**

  Paper based documentation is most simple to use as threat modelling concepts are visualized using regular notes, diagrams and hand-drawn tables. It has clear advantages of requiring no development and a low learning curve. Unfortunately, this approach is not flexible and updating information is rather complicated. Data persistence and handling is also questionable as of any paper document as information can be easily lost. Moreover, penetration process is completely separated from the threat model.

- **Software based threat modelling tools**

  Software based alternatives for threat modelling are numerous. Microsoft Threat Modelling Tool is used as key tool in Microsoft Security Development Lifecycle(SDL)[6]. Similar tools are well tested and widely used. Their flexibility and adaptation may be advantageous in most cases but that also creates a learning curve and requires adaptations to be used with a particular threat model. As in the previous solution, there is no concept of integration with penetration tools.

- **Penetration testing using a bundle of tools**

  Penetration testing frameworks like Metasploit are incredibly useful and widely used across cyber security industry. Metasploit framework has a large variety of tools and even covers report generation. Similar solutions could be used in combination with previously mentioned threat model documentation techniques in order to perform a full system analysis.

As it appears, there are suitable solutions for penetration testing and threat modelling already available on the market. Unfortunately, it is rather complicated to find a solution that would directly link penetration testing tools with threat modelling methodology. To the writers best knowledge there are no such tools specifically developed for Internet of Things systems analysis. On the other hand, existing tools could be modified or used together in combination to achieve a similar result.

6.2 Project scope

The project prototype is intended to be a convenience tool to assist testers in performing penetration testing. The tool set is not supposed to run penetration tests on its own as it is not an automated penetration test runner.

The tool purpose is to design a way of structuring threat model data in a certain form which it could then feed to any existing penetration tool receiving meaningful output. Information visualization is not the top priority of this project as structured data can be displayed in any way needed using a range of applications.

This project is specifically targeting IoT penetration testing but due to the shared variety of possible IoT technology variations it is impossible to support all devices. Therefore, the tools included in this prototype are only basic tools required to demonstrate its functionality. There is no intention to make it compatible with interactive tools as well.

The graphical user interface is expected to be minimalistic giving priority to usability and functionality instead of aesthetic design.

6.3 Requirements

6.3.1 Functional

- Functionality to add and remove new security tools to and from the application

- Contain an initial proof-of-concept set of tools
- Provide functionality to scan local network, map network infrastructure and identify its components
- To be able to chain one tool output as another's input
- Support at least 2 communication protocols: WiFi and Bluetooth
- Provide functionality to build an IoT penetration testing threat model
- Process threat model content to provide suggestions for penetration tests
- Generate structured penetration testing data
- Create graphical user interface for the application

6.3.2 Non-functional

- Compatible with the most popular Linux distributions
- Packaged to include all the required dependencies and is portable with little setup
- Must provide its user with feedback within 2 seconds of user interaction
- Must be able to identify at least 25 unique devices on the network

6.4 Possible approaches

6.4.1 Extension of an existing penetration testing framework

A usability-oriented plugin designed for a pen testing framework like Metasploit may be capable of meeting most technical requirements. A solution like this would be closely linked to the framework of choice and its implementation. Project extendability would also be questionable as it could only accompany the tools and functionality provided by the framework. The main drawback of such approach is that, to the writers best knowledge, there are no IoT specific penetration testing frameworks currently available; thus, it is unlikely that a single, general purpose (or web specific) penetration framework would contain a wide range of tools needed for this purpose.

6.4.2 Web based solution

A web-based solution has also been considered. A local web server would run on the users machine hosting a website which the user would interact with the system through. If this approach is taken, differences between browsers and cross-compatibility would need to be taken into consideration. Moreover, it would significantly slow down the development process due to the lack of the writers familiarity with web programming and suitable technologies.

6.4.3 Desktop application

A desktop application would provide much freedom for adjustments and would only depend on the libraries used, which can be shipped together with the application. Depending on the choice of a GUI library, a native prototype look can be achieved without any effort. The writer is familiar with desktop-based technologies, therefore, the implementation stage is expected to be easier and more functionality may be provided within the same period of time. A desktop application would require installation, however, that is no different from web-based or extension approaches. On the other hand, a web-based or plugin-type solution may provide a more aesthetic design.

6.5 Chosen approach

After considering the listed pros and cons, it has been decided to build a desktop application due to the freedom of modifications, past project experiences and solution portability.

In addition, a high-level language supporting rapid development and native to most Linux platforms has been determined to be the best suited for the task.

Therefore, a desktop-based Python solution has been chosen.

7 Design

7.1 Tools and techniques

7.1.1 Tools

- **Github**: the source control platform used during the whole project development. Commit history, auto-generated progress statistics, progress tracking.
- **Visual Studio Code**: editor of choice, syntax highlighting, integrated debugger and unit test executor.
- **virtualenv**: Python environment isolation tool. Separates the development environment from the global machine environment, mitigates indirect permissions, dependencies and version incompatibility issues.

- **pip**: Python package installer. Keeps track of project dependencies, simplifies installation and deployment.
- **Lucidchart**: web-based tool for creation of UML diagrams and drawings. Used for planning and documentation purposes.
- **pylint**: source-code bug and quality checker for Python following PEP8 style guide[8]. Highlights pure code practices and imposes the use of good coding practices.
- **autoDocstring**: Visual Studio code extension for Python comment generation. Help to keep consistent commenting style and contributed to good coding practices.
- **Qt 5 Designer**: easy-to-use graphical user interface tool for designing and building Qt UI for the static part of the application.

7.1.2 Libraries

- **pytest**: Python unit test framework. Extensively used during the whole prototype development process.
- **jsonpickle**: Third-party Python library used for exporting threat model as structured data.
- **ruamel.yaml**: Third-party Python library used for parsing Yaml files. Yaml files were used to describe tools included in the tool set as well as providing 'caching' for the threat model entries.
- **PyQt5**: a Python binding for a well-known Qt v5 cross-platform C++ libraries set. It provides native look across common platforms and is used for the visual part of the projects.

7.1.3 Techniques

- **MVC pattern**

  Model-View-Controller architecture is used to separate the application data from its visualization. Due to the inherited modularity of the project - combination of a threat model and a tool set - two separate instances of MVC pattern are used.

  The first MVC instance represents the threat model. The overall threat model data object is linked with the parent threat model Controller which contains the narrow purpose Controllers. Each Controller object in the parent Controller is linked to corresponding View object. View objects represent separate parts of threat model GUI.

The second MVC instance is needed to structure the tool set. There is a generic object containing all the separate penetration tool information. Each independent penetration tool has its own Model representation. All penetration tools in the tool set have generated graphical user interfaces. These GUI's are created together with linked controllers by the tool Model objects in accordance to the tool data stored in those Models.

- **Facade pattern**

  Facade pattern is used to hide internal system complexity providing a simpler interface to use. In the project it is used to mask threat model internal relations providing a general "Threat Model" object to work with. The internal structure of a penetration tool object as well as the process of parsing its configuration data is also encapsulated.

- **Interpreter pattern** (just defining language and parsing in recursively)

  Interpreter pattern is used while parsing penetration tool configuration files. Complete majority of pen testing tools can be run from the terminal. Unfortunately, various tool syntaxes tend to have fundamental differences. Therefore, one part of configuration parsing process is to interpret unique tool syntax which may also contain recursive patterns.

- **Observer pattern**

  Qt graphic components can make use of observer pattern to notify linked devices of different events. In Qt this pattern is called "Signal and Slot" mechanism. Signals can be "emitted" in response to system or custom actions and connected functions - Slots (can be enclosed in other objects) will execute in reply. It is used across the application to create dynamically linked and responsive GUI.

- **Object pooling**

  As the application uses threat pooling technique it supports parallel execution of multiple penetration testing tools. This way application main event loop is separated from individual tool execution.

- **Serialization**

  Serialization is used to preserve information kept in the threat model between work sessions. Threat modelling projects can be saved, closed and re-opened. Due to the way Python serializes objects the tool set implementation can be altered (e.g. a new version is released) still maintaining compatibility with threat model files created by older software versions.

7.2 Application structure

7.2.1 High level design

High level project structure is presented in figure 1 below.

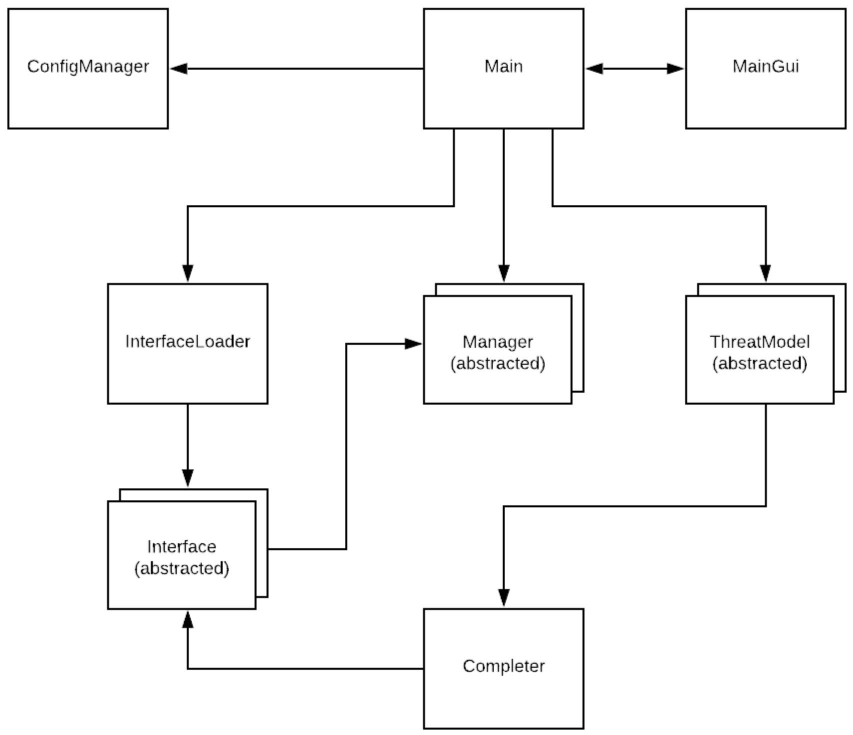

Figure 1: High level project structure

It consists of two larger and four smaller parts. The "Main" and "MainGui" components represent application parent classes. They are responsible for configuration, initialization and start up. During startup phase "ConfigManager" is called to load global system configurations such as interface folder location and cache configuration. The "Interface Loader" object is one of the central parts of the application. Its purpose, as the name suggests, is to parse penetration tool configuration files and create tool "Interface" instances. "Manager" object supervises penetration tool execution and presents output. "Threat Model" object hides internal threat model complexity and governs its display. "Completer" object is used to link threat model data with different penetration tools and their options. It generates input suggestions during pen testing sessions. "MainGui" object loads the main parts of the GUI structure and controls threat model display.

Project layout diagram is provided in appendix C.

7.2.2 Penetration tool interface

Simplified penetration tool interface structure is presented in figure 2 below.

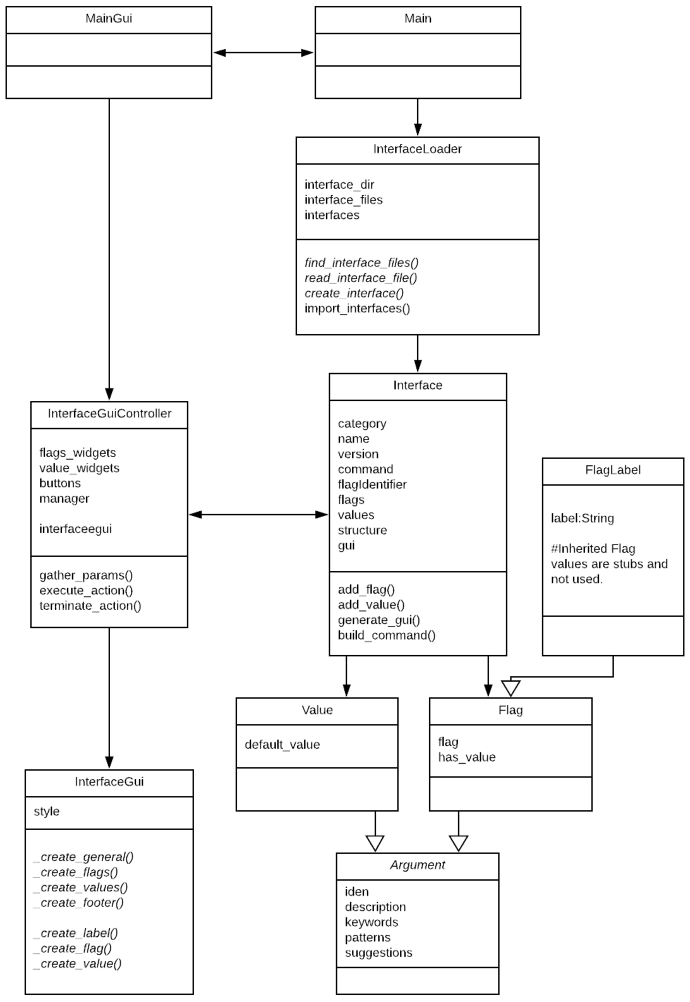

Figure 2: Simplified penetration tool interface related project part

"Interface Loader" object is responsible for finding, parsing and storing added pen testing tool interfaces. Individual penetration tools are described using their configuration files; file structure example can be found in appendix **??**. Configuration files define tool functionality and the way to map command line options to GUI entities. "Interface" objects are individual penetration tool representations in this project. "Interface" instances encapsulate overall penetration tool information required for its use. Pen tool arguments are represented by the "Flag" and "Value" instances which are held in the "Instance" object. "Flag Label" class is used to group "Flag" instances and as a visual marker in the GUI.

Configuration parser design is recursive; thus, "Interface Loader" only checks the general configuration file structure. Similarly, "Interface" upon receiving data is only responsible for its attributes, tool arguments are parsed by "Flag", "Value", "Flag Label" and "Argument" classes accordingly. Distributed parsing reduces complexity and splits up responsibilities making verification and testing easier.

Interface GUI is structured following MVC pattern and is automatically generated from parsed attributes. This design supports nested tool parameters without any additional configuration. However, out of projects scope, Qt library provides a markup language similar to CSS. Therefore, due to the recursive nature of the GUI design, only insignificant modifications would be needed to provide a neater user interface.

7.2.3 Tool execution and output

Simplified penetration tool execution and output display diagram is presented in figure 3 below.

Individual pen tool (from now referred to as 'interface') execution is handled by the "Manager" object. Every interface GUI has a reference to the global "Manager" object. Upon every new interface GUI addition, a "MainGui" passes an "Output function" to the "Manager". The "Output functions" are mapped to individual interfaces inside the "Manager" object, after the interface execution they are called to display the results. Due to the variety of tools available and their visual output differences, the display functionality has been abstracted out from the system. Therefore, it is no complicated to extend the prototype with other ways of displaying the output.

The "Manager" upon being given a command line string to create a "Worker" instance which is responsible for its execution. "Workers" start a new subprocess to run the command in the terminal. They can also receive updates on the process execution using "WorkerSignals". "Manager" can be asked to terminate a command execution which by itself will send the termination signal to an appropriate "Worker" object. When a "Worker" finishes the assigned task, it will use the "callback" function to notify the caller interface about task completion.

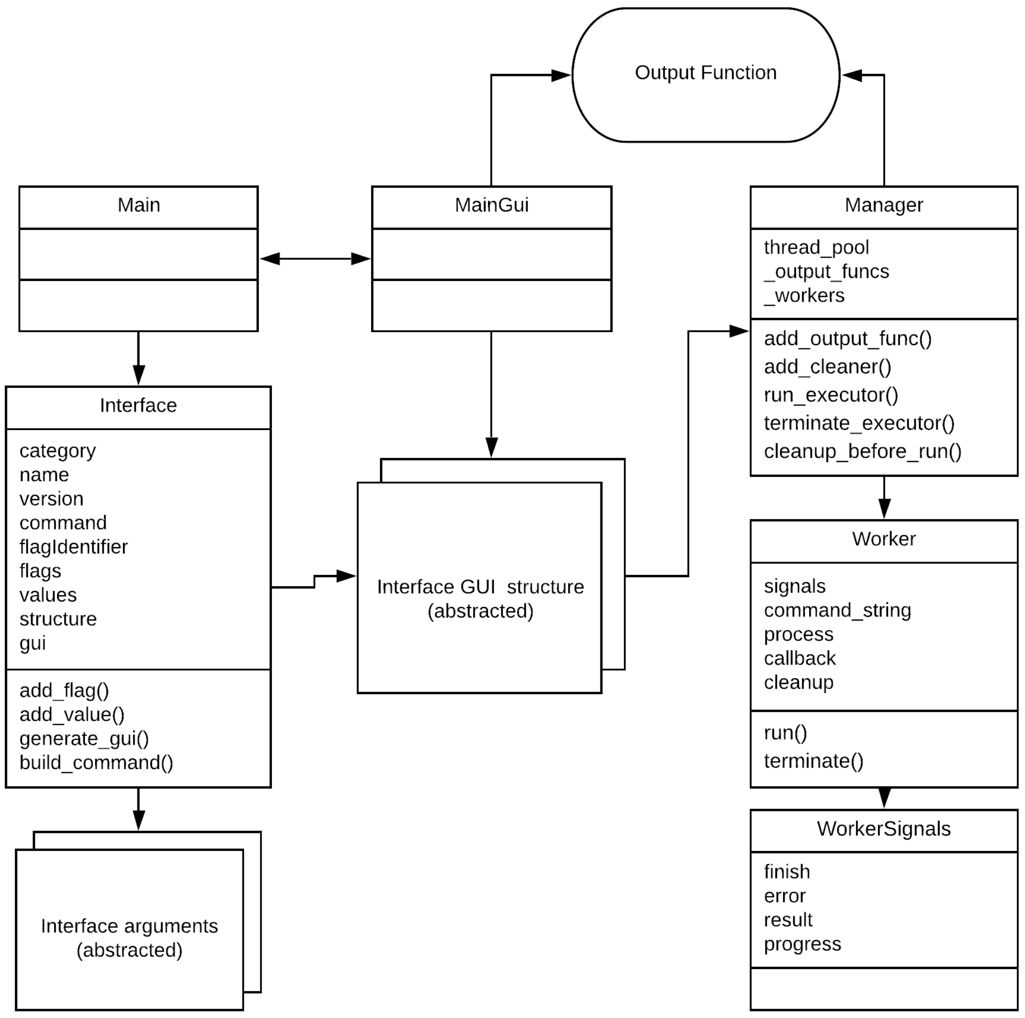

Figure 3: Penetration tool execution

7.2.4 Threat model representation

Simplified proposed threat model implementation diagram is displayed in figure 4 below.
IoT threat model described in "IoT Penetration Cookbook"[19] is implemented in the application threat model section. The implementation uses MVC pattern and follows Single responsibility principle[11].

The threat model internal relations are abstracted using the "ThreatModel" object. As covered in the 5.4 section, every IoT system is composed of "Assets", which are physical or virtual entities of the system. Every "Asset" can employ a number of technologies, in this design it would be more appropriate to say that unique "Technologies" can be "used by" multiple "Assets". Every system contains multiple interaction points, here referred to as system "EntryPoints". Each "EntryPoint" is present in an "Asset". Potential system vulnerabilities are represented by the "Threat" objects. Each "Threat" is linked to a system "EntryPoint" and possible by exploiting some "Technologies" weaknesses. The individual "Threats" are then ranked using a ranking system - in this case DREAD, represented by the "DreadScore" object.

View and Controller parts are divided into multiple segments responsible only for the specific bit of GUI functionality. Threat model elements are organized using a tab menu where each tab content is generated by a different View object. Information displayed in the separate tabs is interconnected, therefore, the observer pattern is used to keep it up to date. To make user interaction more convenient, the application remembers previously entered items making the GUI more user-friendly. The item 'caching' takes place on system level, therefore, items from one threat model can be easily imported to another. Threat model front-end and back-end parts are connected view of the parent "ThreatModel-Controller" and ThreatModel" objects.

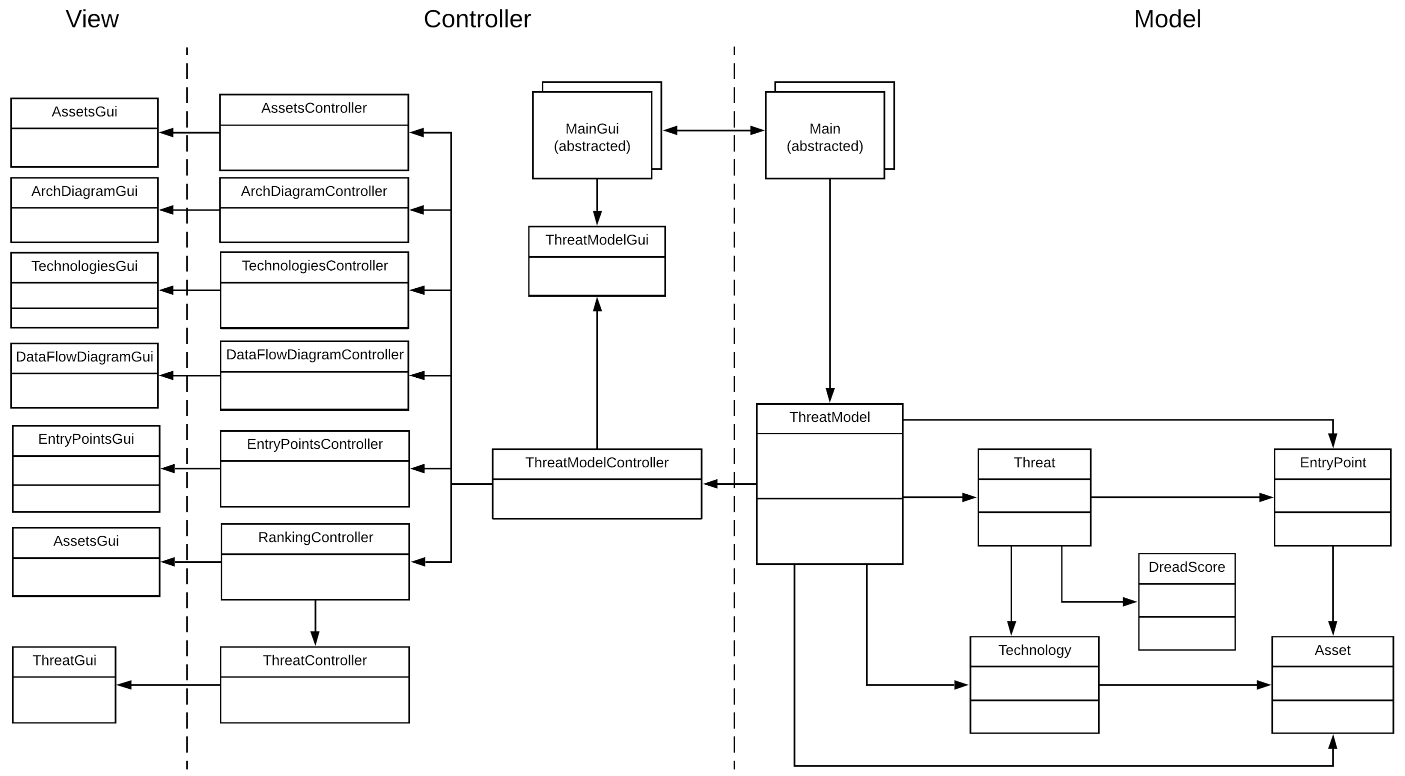

Figure 4: Threat Model structure

7.2.5 Suggestion generation

Pen tool argument suggestion process diagram is presented in figure 5 below. "Completer" object scans threat model data for specific words and phrases that match interface argument patterns. This functionality is provided as a convenience mechanism for the user linking threat model data with penetration tools included in the tool set. The "Completer" can only link keywords to assist pen tester work but at the current state cannot suggest what kind of penetration test needs to be used.

Interface configuration files can optionally contain "keywords" and "patterns" for every tool argument. These two sets of values can then be interpreted by "Completer" and matched to user entered values in the threat model. In order to separate threat information, completer requires a specific threat to be selected from the threat list. This way information unrelated to that threat is filtered out. "Completer" works by scanning every "Threat" object attribute, then the "EntryPoint" object which is linked with that "Threat" instance, then the "Asset" object of the mentioned "EntryPoint", and finally "Technologies" suspected to be used by that "Threat".

In the pen tool GUI suggestions are presented as text completion drop down list, which helps to quickly enter values for the selected test. Only tool arguments that require the user to enter information by hand may have suggestions generated; Boolean type command line arguments cannot have any values, therefore, suggestion model cannot be applied to them.

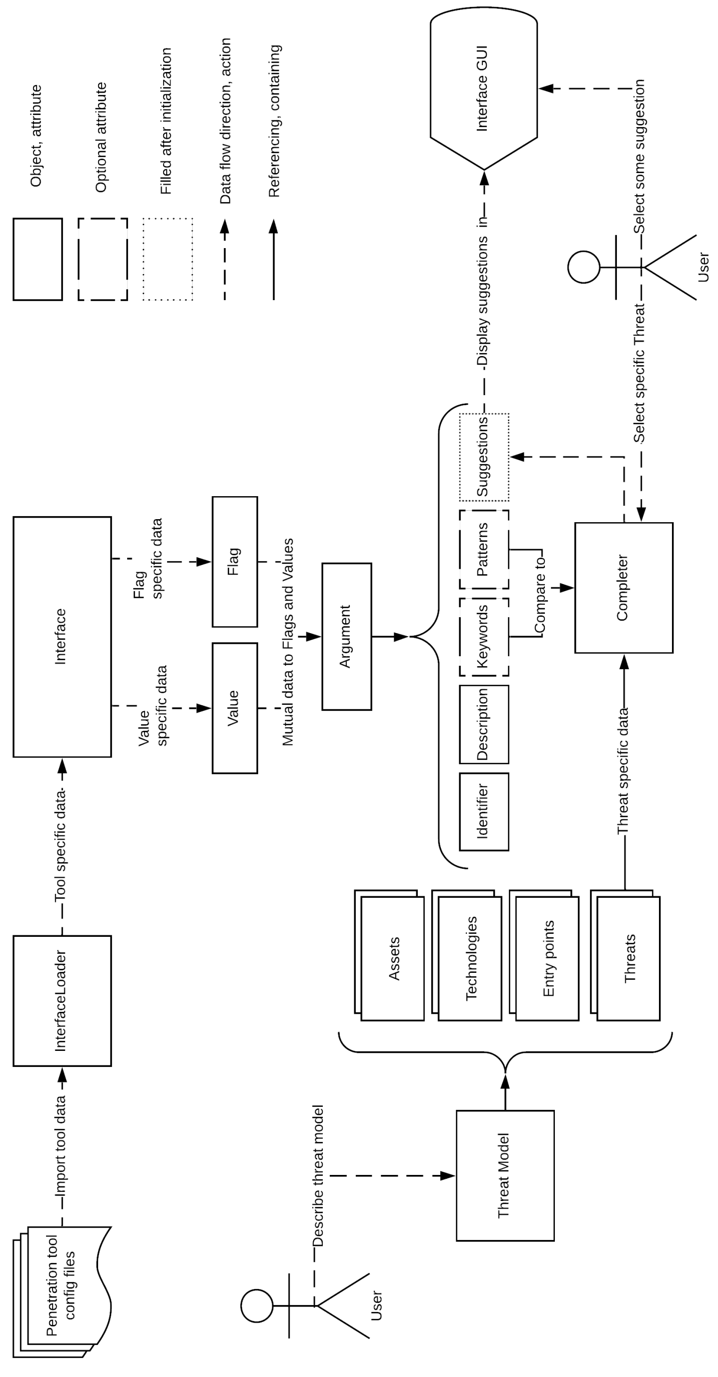

Figure 5: Suggestion generation control flow

8 Implementation

This section of the report covers project prototype development details more in-depth and puts emphasis on specific implementation decisions. It also covers the list of well-known penetration tools included in the tool set.

8.1 Development process

8.1.1 Development

Project prototype development has been scheduled as a continuous process during the last months of the project allowing a lot of freedom for alterations. It has been decided to follow Agile development methodology dividing tasks into Sprints and using supervisor meetings to present that Sprints results. An in-depth, project management discussion is covered in section 10.

Test-driven development (TDD) practices have been used in order to complete large amounts of robust code in limited Sprint time periods. TDD has been used for development of the whole project back-end functionality and to test Interface GUI generation, for the front-end part has been tested using conventional testing techniques. Testing process is described in detail in section 9.

Greatest effort has been put to follow Python PEP8 coding conventions to achieve maintainable, structured and standardized codebase. Pylint and autoDocstring VS Code extensions have been particularly useful for this goal.

8.1.2 Debugging

In the case of unexpected program behavior, VS Code built-in debugger has proven to be particularly useful. Its stepping through the code and break point features have been used extensively during the development process. Built-in Debug console has also proven to be convenient in order to verify program object states and the case of exceptions or break points.

8.1.3 Deployment options

After the implementation is finished, the prototype can be deployed as a Python package using dependencies management system as PIP. Installation of project dependencies would be taken care of by PIP. In order for the prototype to use tools included in its tool set, individual pen tools would need to be installed independently in the system. Bundled program installation is one of the future work features discussed in the section 11.3.

8.2 Tool included in the tool set

There are nine tools included in the tool set prototype. These tools have been chosen according to five areas of IoT Penetration testing[19]. The tool set, as mentioned before, is not by far complete, however, demonstrates that the prototype supports a wide range of pen tools.

Tools are grouped in sections based on their use. Only Hardware hacking section is not covered by this tool set as even if there are software tools used for Hardware related hacking, it is likely to be architecturally specific and require additional physical devices or modules. Remote access section is also present in the set as its tool functionality is rather distinct compared to other Web application or Wireless sections tools.

- Firmware

  1. **binwalk** - is a tool for automated firmware extraction from the device binary image. It provides a wide range of configurations and supports different image formats.

- Web applicaiton

  2. **sqlmap** - is an SQL database penetration tool. It has a powerful search engine and a range of switches configuring its execution. The tool is useful if an IoT system is suspected of storing data in a database.

  3. **wfuzz** - is a web application brute forcing tool. It can fuzz GET and POST request parameters as well as web resources that are not linked.

  4. **dirb** - is a web content scanner software. It is similar to wfuzz but it uses fuzzing to discover hidden web resources and left-over files.

- Mobile application

  5. **apktool** - a tool for reverse engineer closed-source Android apps. It is capable of rebuilding original file structure and decode some of the initial application resources. It is intended to be used for Java based mobile applications.

- Remote access

  6. **hydra** - hydra is a multi-process login cracker. Hydra supports a wide range of communication protocols and can be customized to try several different approaches during its execution.

- Wireless

  7. **hcitool** - is a tool for configuring Bluetooth connections. It supports Bluetooth device discovery, most popular Bluetooth communication

protocols and can be used to send specific commands to the Bluetooth device in order to pen test the communication.

8. **nmap** - is the most famous network discovery and port scanner currently in the market. It is highly configurable to the point where the whole network infrastructure can be mapped and tested by just using nmap.
9. **sdptool** - is a Bluetooth service discovery tool. It can connect to individual discovered Bluetooth devices and retrieve information about its advertised services.

8.3 Third-party code used

Apart from the libraries, which are listed in the section 7.1.2, third party code has been used in a few insignificant places in the code base:

- In the file *line.py* for custom Qt GUI element implementation.
- In the file *vtabwidget.py* for rotating the Qt GUI tab container titles.
- In the file *manager.py* for implementing *WorkerSignals* communication model.

In all the cases, appropriate headers have been used to clearly indicate third-party code.

8.4 Implementation details

This subsection covers some important implementation details that were not covered in the Design section.

8.4.1 Application preferences file

The program has a global preference file which is used to define application level settings. Following Python conventions, such information is stored in an '.ini' file, which is then parsed and maintained with built-in Python "configparser" library. In the case of complete loss of the configuration file or one of its mandatory fields, a new '.ini' file is populated using reasonable default values.

8.4.2 Interface file structure

Penetration tools used in the tool set are referenced via their "interface_x.yaml" files. Each file has mandatory and optional fields. An example interface file is shown in appendix D.

The file is in YAML format which was chosen due to its simplicity, readability and compatibility. As these interface files must be configured by hand for every penetration testing tool, intuitive structure and readability was a priority. If, for example, XML format was chosen, it would be much more complicated for the human to compose configuration files.

It is important to note that tool flag and value lists can be empty and can also be defined recursively (e.g. flag containing optional flags). Configurations described in YAML format are easy to add and auxiliary fields would just be ignored. Adding a new tool to the system is as simple as creating a new interface file from the example and adding it to the appropriate folder.

8.4.3 Graphical User Interface functionality

Care has been taken to make the application GUI intuitive and simple to use. The application takes extensive use of tabular structures. Penetration tools are grouped according to their purpose using separate tabs. Threat modelling project part uses two-level tabular structure to group related functionality. Every threat model entry can be edited, duplicated and deleted. Threat model items (e.g. entry points) are cached on the application level, thus, they can be reused in other projects, cache can easily be cleared for separate threat model item groups (e.g. technologies). Threat models can be loaded, edited and saved again; appropriate warning messages are thrown while trying to close threat model with unsaved information. Threat models can then be exported as json files for data visualization. Appropriate native hotkeys can also be used for these operations.

8.4.4 Scalability

Prototypes scalability is two-fold. On the one hand, the application can easily import as many tool interfaces as provided and load threat model files as big as needed. On the other hand, large files were not taken into consideration during the development process and no special action has been taken to parallelize or divide any of the tasks in any way. Effort has been made to use native GUI element as scrollbars to display large information sections and separate GUI functionality using tabs. Nevertheless, it may be inconvenient for the user to navigate through large numbers of tools and threat model items.

As described in section 7.2.3, penetration tool execution and output generation are parallelized using functional programming, therefore, tool execution scalability is only dependent on users setup.

9 Testing

Project testing was performed in two ways: software testing during Test-Driven (TD) development and a tool use case analysis.

9.1 Software testing

As it has been mentioned before, application testing process has been integrated into software development by following TD development method. Full test log can be found in appendix E. There is a total of 109 separate test cases being tested.

9.1.1 Unit testing and Integration testing

For each individual piece of software functionality an appropriate unit test had been created before starting implementation. Then a function passing that test was written. Afterwards, more test cases were added to test alterations in control flow, finally a code was refactored to comply with all test cases.

Following this workflow, appropriate pieces of code were tested during their development. It also matches with individual Sprints; therefore, fully tested pieces of software were presented during each after-Sprint supervisor meeting.

Software test log in Appendix E is a mixture of Unit and Integration tests. Due to the TD process, their distinction is not obvious as some cases evaluate a combination of object functionalities (e.g. parsing and creation of interface files) and others only specific functions (e.g. adding a new Asset object to ThreatModel).

9.1.2 GUI testing

GUI prototypes were tested using manual testing process during development. Program GUI was not a priority for this project and the testing that has been completed for it was not as in-depth as for the back-end software part. Program GUI functionality has been checked again during the Usability testing phase.

9.2 Usability testing

Usability testing demonstrates prototype usefulness in close to real world system setup. The goal of it is to go through the whole pen testing process using the tool as an end user would. This section describes the tested IoT system, concluded penetration testing results and evaluates how practical the tool is.

The IoT system chosen for testing is a home alert system built for a different module. It contains a variety of distinct network solutions and technologies (e.g. Bluetooth LE, WiFi, cloud and mail service); thus, simulating a large real life IoT network. For this case a "grey box" testing approach had to be employed. Ideally, penetration testing using the "black box" approach would have been preferred as then real first-time system analysis using the IoT penetration tool prototype would have been achieved. Regrettably, a large part of the alert system was developed by the author of this report, therefore, some system insight knowledge could not be denied.

The penetration testing started based on the implemented IoT penetration testing methodology, followed by appropriate tests and threat model updates. Screenshots of this system penetration testing process can be found in the Appendix K and the saved file is included in the submission. Distinction between system assets, technologies and entry points as well as the connections between these entities helped to identify the weaknesses of the system. Then Nmap, dirb, wfuzz, sdp, hcitools and hydra tools were used respectively to assess the potential threats. Neither firmware, nor mobile application analysis took place as they were not needed in this case. The alert system pen test has shown several significant security issues mostly due to the use of HTTP instead of HTTPS protocol and weak to non-authentication. Physical system component vulnerabilities could also be exploited to disrupt the workflow.

The usability test has shown that the prototype can be used to test real life IoT systems. The GUI is intuitive, responsive and functional, although, not well-styled. The penetration tools included run as expected, however, it is advisable to use the prototype as a root user because some parts of the tools that are included, require to be run with elevated privileges. The interactive keywords transfer from threat model to respective pen tool fields works as intended. Overall, the IoT pen testing tool works well and is suitable for the task.

10 Project management

This section covers time and workload management during the whole project development period. It explains important project planning decisions, covers the strategies used for time planning and work distribution. It also compares the project schedule from the Progress report with the final project schedule. Finally, it evaluates project management and explains the deviations from the initial plan.

10.1 Early prototyping period

At the end of the first semester exam period, it has become clear that before proper development could be started, some degree of experimentation is required. Firstly, experiments to get familiar with Qt development patterns, functionality and convenience software were needed. Then, research into data structure parsing, finally settling with yaml data format. Thirdly, some time was needed to test command line string generation and execution in the terminal from Python.

Unfortunately, there was a need for more background research especially in the specific penetration tools that would be most suitable to perform the test for IoT devices. The delays shifted the project schedule a few weeks and postponed the actual prototype creation.

10.2 Agile methodology

A range of Agile practices were followed during the whole project timespan. Those include well-defined sprints, demos, task prioritization, Test-Driven and rapid development. As this project was done by only one person, not all Agile practices even make sense as there was no cooperation between team members involved. In most cases, Agile practices were seamlessly adapted in order to efficiently produce robust software and keep track of the remaining and completed work.

10.2.1 Sprint planning

Sprint task planning took place before each sprint and sometimes had to be altered during the sprint week. The tasks were compiled in a feature-orientated manner, dedicating whole sprints to individual parts of the application. This approach minimized the need for code refactoring in later sprints. Sprint plans were stored and updated using source control, removing the need for dedicated sprint planning programs as there was little to be gained from it in a one-person developers team.

Each sprint would end by a supervisor meeting which conveniently was similar to Agile sprint demo practice. During each meeting sprint progress and completed features were discussed as well as defining tasks for the upcoming iteration. Similarly to a customers demo feature improvements and suggestions were considered and the sprint plan adjusted accordingly.

Appendix F includes formatted sprint plan according to MoSCoW prioritization. Reader may notice that different sprints lasted a varied number of days, this was needed to accompany other assignments and varied workload. Some tasks (marked red) were not completed on time or moved to the next sprint.

10.2.2 MoSCoW prioriterization

Using MoSCoW requirement prioriterization technique each Sprint task was ordered according to the necessity and potential product value. Every sprint task was categorised into four groups: Must, Should, Could, Won't (highly unlikely) on the day of the supervisor demo meeting. This structured approach kept the author's focus on the most important features assigned for that Sprint. A simple list of prioritized tasks provided an uncomplicated way of following sprint progress and the remaining work.

10.2.3 Github source control

Github source control platform has been used frequently to log project progress and development. Following rapid programming principles, it was aimed at frequent and single functionality commits to aid debugging process. Git managed notes also served as a sprint log between development cycles. Conveniently, Github also generates graphs illustrating project development.

Project statistics can be found in Appendix G. Commit history graph and the code frequency graph shows rapid development period of five sprints from the end of February till the beginning of April and the early experimentation phase before it. Commits in the last weeks of April are due to the project report being written.

10.3 Test driven development

From the start of the project Test-Driven development (TDD) technique has been chosen as the most time and resource efficient way of delivering robust code. Due to the delay suffered from the early prototyping phase, development schedule had to be pushed into a narrow window with little to no time for other setbacks. The code had to be developed efficiently, according to Sprint plan and fully tested.

As each Sprint was finalized by a supervisor meeting during which working code had to be provided, a high risk of (un)intentional cutting of corners was possible. TDD approach was ideal for such setup in order to ensure proper software testing and delivery of working code.

10.4 Project schedule

Gannt charts illustrating the expected and the actual project schedules are included in the Appendix H. As it is apparent, due to miscalculation and confusion between report submission and the viva dates, the actual project development had to be shortened by a few weeks; that resulted in moving the report writing into earlier weeks. The need for additional background research and longer prototyping periods forced the development sprints to be pushed back a few weeks and contracted into a smaller time window. Nevertheless, the expected and the actual schedules do not differ significantly as most of the planned activities were completed in the predicted time slots.

10.5 Risk management

The project risk table is included in the Appendix I. The table contains the possible risks and their mitigation techniques. It can be concluded, that there were no unexpected risks during the project development stages. Prototyping and research phases took longer than expected first, however, the situation was resolved via the applied mitigation techniques.

10.6 Project management evaluation

Overall, the project time and resource management were backed up by the methods and techniques used in industry following Agile development framework. Each of the project phases had been allocated specifically loose time frames leaving enough flexibility for alterations. In general, the defined schedule was met with minimal changes.

Prototype coding tasks were divided into reasonable sprints and prioritized using MoSCoW technique. Task prioritization helped with the project progress tracking and further scheduling. Development and testing process integration ensure that the code is robust and self-encapsulated.

Project risk table and contained mitigation techniques allowed for straightforward situation adjustments minimizing the time loss. Schedule alterations only had to be made for longer background research and prototyping periods.

10.6.1 Cutting corners

In order to comply with compressed programming schedule, some elements had to be omitted. In particular, prototype GUI element style and design were not taken into consideration. Additionally, a requirement for chaining individual penetration tool execution had to be skipped (further detail in the Evaluation section).

Although prototype is self-explanatory and intuitive, there is no documentation provided as it had to be dropped out due to time constraints. The prototype deployment and compatibility with other systems is also not fully tested, despite the fact that there are no reasons for it to be incompatible.

11 Evaluation

11.1 Requirements evaluation

The prototype provided complies with all but one previously defined functional and all but one non-functional requirement.

Importing or removing penetration tools from the tool set is uncomplicated by moving interface files from the dedicated folder. The proof-of-concept tool set contains various tools including the ones for LAN network scanning and Bluetooth communication protocols. The implemented threat model is designed to be updated with new data from the penetration test runs, as well as mapping appropriate keywords from the threat model to the individual penetration tools. After testing is completed, the threat model together with the included testing results can be exported as structured; JSON output to be used later or displayed in a separate report. The functional requirement which was not met is the tool output chaining. This specification had to be refused due to its complicated nature involving universally supporting parsing of penetration tools output and its limited cases of use.

The tool has been tested and run on Ubuntu OS using Python and Pip package manager which guarantees its compatibility with other Linux operating systems that support the required dependencies. It also implements multithreading, thus, moving all the computationally intense tasks away from the main program threat and ensuring quick program response. The prototype does not scan for devices on the network itself but employs well-known pen tools to complete these tasks, it shifts the responsibility to Nmap and Hcitool implementations which can perform scans on huge networks without any difficulties. The unfulfilled, non-functional requirement was the complete encapsulation of the tool in one portable program. Due to a shorter development period and the relatively low additional value of this feature, the requirement was not completed. The prototype packaging is somewhat finished as third-party dependencies can be installed automatically using a package manager.

The final result is in accordance to the project Brief included in Appendix A and the project goals raised at the start of development process are met. A slight shift from the project brief goals is that the final prototype focuses on the use of IoT penetration methodology and not just penetration testing.

11.2 Usability and testing

The tool usability has been evaluated during an actual IoT alert system penetration testing. The system was tested using ”grey box” method which combines internal system knowledge with the outside attacker methods. The IoT penetration tool proved to be usable and suitable for this kind of penetration testing, nevertheless, additional work in tools GUI design would be welcomed.

The tool has been tested with a combination of unit and integration tests with an emphasis on the tools back-end functionality. The front-end and GUI testing has been done manually and by doing the evaluation of the alert systems.

11.3 Future work

A few parts of the IoT penetration tool could be improved in the future.
At the current moment, the system can pipe penetration tools output to any destination, however, only the most basic terminal-like display is included. A more user-friendly way of presenting penetration tools output would greatly improve the usability of the tool. During the project development it was decided to implement a generic approach as individual penetration tools use their own output formatting and may support other forms of data representation. As the tool is aimed at supporting any command-line based penetration tool, a generic approach which can be extended to other formats, makes more sense.

For the prototype to include advanced tool functionality GUI design has been neglected and a properly formatted GUI could make the tool more readable. As the tool GUI design is not optimized some options or fields are not displayed in the most efficient manner. The same could be said for the colour scheme and text font styles.

Effort could potentially be made to expand the current tool set by writing interfaces for a package sniffer or for penetration tools used in other areas e.g. firmware analysis. As the project requirement was to provide an extendable penetration tool platform and a proof-of-concept tool set, only the basic cases of use are covered.

Some additional work could also be done for the tool to convert structured threat model format into a user-friendly interactive report displaying key penetration testing results. The prototype provides structured data which could be represented in any way and in any format needed. Report generation in a particular form would limit the gathered information to a single format that could not be easily reused.

11.4 Conclusion

This project addresses the rise of IoT related cyber security risks and their exploitation. It provides a platform where threat modelling is combined with penetration tools. The tool is trying to fill the void of dedicated IoT penetration testing solutions which could be used by people with little cyber security related experience. It is essentially a highly extendable IoT penetration testing platform closely linked with IoT threat modelling methodology. The two are coupled in order to guide the user to properly structure and document the penetration testing process. The proposed "IoT Penetration Testing Toolset" could also be called a penetration testing assistant as it helps the user to structure the system evaluation process. It is hoped that the tool would assist in creation of more secure consumer and industry level IoT systems.