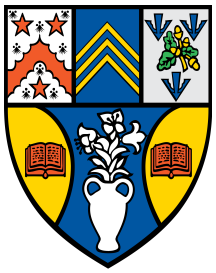


Visual Malware Analyser

Qubytes: Jack Bowker, Kayleigh Gall, Sam Heney and Mairi McQueer

CMP311 - Professional Project Development and Delivery

BSc Ethical Hacking

21st April 2020

Abstract

This white paper details the process of the development of a Visual Malware Analyser. This was requested by lecturers at Abertay University, Ethan Bayne and David McLuskie, to assist with teaching students on the Ethical Hacking course. The Visual Malware Analyser (ViMA) provides an easy to understand visualisation of a worm propagating through the "Hacklab" network.

The original brief breaks down into three components; a web page to visualise the propagation of malware, along with a worm and anti-worm. Although this project is centred around ViMA, the worm is important as it provides the data being displayed by the web page. The final project has been designed so that these components are non-essential and the visualiser can still perform it's intended role by being fed mock data to display.

Qubytes effectively used the prototyping methodology to implement each of the required components while adjusting development to what the clients required. As the product was developed the team discovered new information which then went on to inform future development cycles. As well as being effective for adapting to client feedback, this methodology proved to be extremely useful for discovering and resolving development dependency issues.

The ViMA web application displays the network's infection clearly and uses modern JavaScript techniques to fetch and display the data in near to real time, providing information about each infected host. Interactivity is a prominent feature of the visualiser, with the user being able to click on a specific host to find out more information about that host, as well as highlighting that host's IP address anywhere it shows in the application.

Both the worm and the visualiser web application make use of an API, which was developed and is run within the Django project along with the web app. Django REST Framework was used to simplify development of the API, allowing the team to focus more on the design and functionality of the API rather than worrying about implementing basic functionality. The API is also completely independent of any other part of the project, meaning that it could easily be reused, or used by other developers, for future projects.

Django was exceptionally beneficial to the project as it also used Python, reducing the number of programming languages the group had to be proficient in.

Overall ViMA is easy to use and clear to read, making it perfect for providing a practical example of worm traversal for Ethical Hacking lectures, thus fulfilling the brief given to Qubytes.

Contents

1 Introduction

1.1 Background

Malicious software or malware, has existed since the late twentieth century and common types include; worms, trojans and rootkits. These are all examples of software that perform harmful activities on a target device in order to disrupt or damage an organisation, steal private information and or for monetary gain.

Worms are widely considered the oldest form of malware and in 1971 Bob Thomas, a researcher at development company BBN, created what are widely considered the first worm and anti-worm, Creeper and Reaper. Creeper was designed to infect machines, leaving the message "I'M THE CREEPER : CATCH ME IF YOU CAN.", and then spread to the next available host repeating this process until it had infected an entire network. Reaper's entire purpose was to propagate through a network in order to find and remove Creeper, making it the first nematode; defined as a virus that is created to combat another piece of malware. *(The Virus Encyclopedia, 2020)*

In march of 2019 the Center for Internet Security (CIS) wrote a blog detailing the most common malware attacks in that month, these ten accounted for around 57% of total malicious software reported. *(Centre for Internet Security, 2019)*

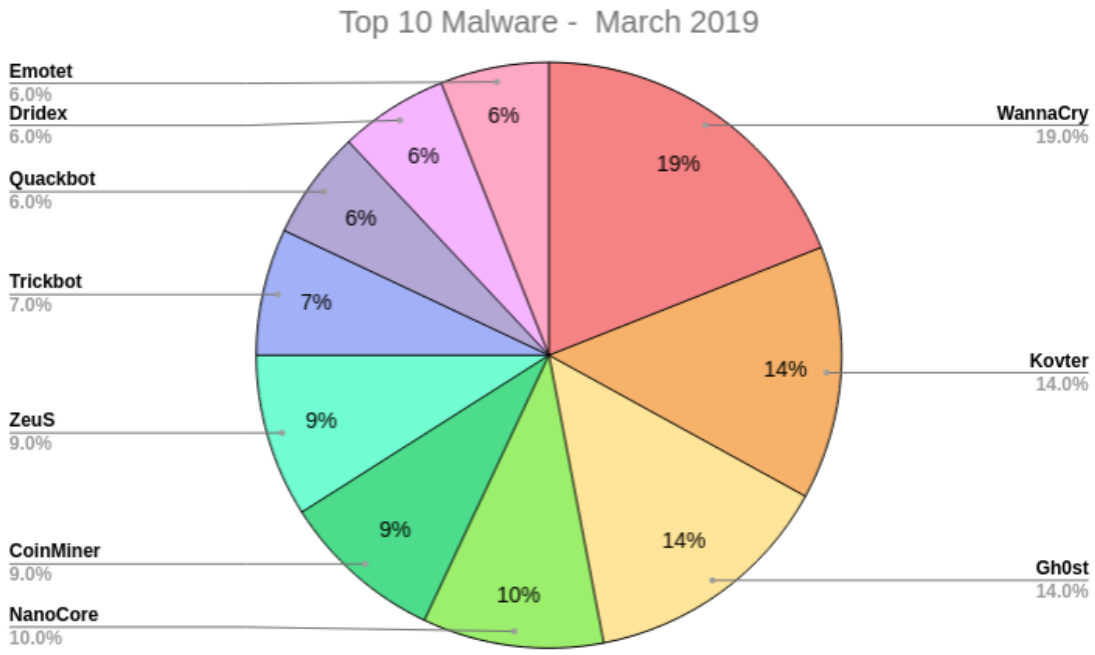


Figure 1: Pie Chart Detailing CIS's Statistics from March 2019

As shown in the pie chart above; 40% of the top ten are worms or a hybrid between a worm and another type of malware, such as ransomware or trojan, which is almost a fourth of total malware reported in that period. Worms for a while were considered outdated and not much of a threat vector thanks to modern anti-viruses and network designs but the wide-scale WannaCry attack in 2017 demonstrated that these vulnerabilities still exist due to the continued use of outdated

Operating System (OS) versions. The virus was defined as ransomware but implemented a worm in order to quickly spread across a network. Over 300,000 computers were infected and it most notably attacked the British National Health Service almost stopping non-essential services for a considerable period. *(Telegraph, 2017)*

This is why it is so important for cyber security students to have at least a basic knowledge of this malware type, since they will be the ones to assist organisations in keeping their software secure or to remove the worm from a network after infection.

There are several different types of learning styles that most people's preferred way of learning falls into; Visual, Auditory, Kinesthetic and Reading/Writing. As a result of this learning style diversity it is impossible to create a lecture that will perfectly suit an entire university class. Qubytes believes by utilising practical examples and visual demonstrations alongside a traditional lecture it is more likely that the information will be better understood and remembered by the majority of the class.

This is why it is crucial for tools and examples to be developed that are able to demonstrate topics such as malware propagation without actually requiring harmful software that could compromise an educational facility's computers or network.

1.1.1 Technologies Used

This project is all designed for and mostly implemented in Abertay University's Ethical Hacking Laboratory, this is referred to as the Hacklab throughout the report.

The worm was implemented using the programming language Python 3.6. Python, released in 1991, is both a very high-level language and a very powerful one with numerous applications. It's lightweight program sizes and ability to run Linux commands make it ideal for projects such as the one detailed in this whitepaper.

Django is an open source web-framework that uses Python and allows for multiple applications to be run from one instance. In this project it is being used to run both ViMA and the REST API.

JavaScript is used in web development to implement complex features that HTML alone cannot handle, such as providing dynamic content and animating images. The JavaScript React framework, which is designed specifically for dynamically updating and displaying content in real time, is used within this project to create the front end of the visual display.

1.2 Aim

ViMA was assigned to Qubytes by teaching staff David McLuskie and Ethan Bayne from Abertay University's Cyber Security Division. The aim was to create an easy to follow visualisation of the propagation of malware through a network, that could act as a learning aide for future Ethical Hacking modules and students. The brief was broken down into the following key requirements; Creating a Visual Malware Analyser that tracks the spread of a worm through a network, along with a harmless worm and anti-worm that will propagate through a white list of IP addresses on the network.

2 Procedure

2.1 Overview of Development Procedure

Within this section the development process of each component of the project will be explained. As the system functions holistically the simultaneous development of each component did end up informing and determining the development of other components. As a relatively abstract example of this, the Anti-Worm can not be developed before the Worm as it makes use of functionality that will be introduced by the Worm. This is a simple example that was obviously accounted for, but as development continued more nuanced dependency issues arose.

Using prototyping as a methodology proved to be ideal for discovering and resolving these issues. Before moving on to the next cycle of development, the prototype created during the current cycle was first reviewed by the team, then demonstrated in a meeting between Qubytes and the clients. This allowed the team to check for any dependency issues that needed to be resolved for the next cycle while also allowing the clients to check if the product was turning out how they imagined. Many times during the development the clients would see some functionality that they would like to be changed, and this was taken into account by the team for the next cycle of development.

Although each of these items were developed concurrently, for the purposes of documenting the procedure as a whole it is more convenient to treat each component as separate entities and describe their development processes individually rather than chronologically. To understand the true chronology of these development timelines, either the commit history on the GitHub repository or the team contributions *(Appendix 4)* should be referred to. *(GitHub, 2020)a*

2.2 Worm

2.2.1 Propagation Method

Initially, in the original proposal Qubytes stated that the worm would be implemented to propagate on hosts running Windows 7, making use of the EternalBlue exploit (MS17-010). In the first week of development this was discovered to be an inconvenient and unsatisfactory solution for the following reasons.

First, the EternalBlue exploit itself is unreliable and dangerous as it could not be made to function consistently and when the exploit worked the target system was put into an unstable state where it could crash and blue screen. Another reason is that there were many difficulties in implementing the EternalBlue exploit in the context of a worm. The team struggled to create a self contained program that exploited the vulnerability and self propagated. *(Check Point Research, 2017)*

Seeing as worm was intended purely to generate data and not for performing malicious activity, the decision was made to not rely on any exploits and instead use Secure Shell (SSH) to propagate the worm. Specifically, a set of SSH keys were generated and copied to the target host using password authentication, navigating the command line dialogue through a scripted virtual TTY. Once the keys were in place, persistent access was easy as no password had to be used.

To make this work the platform was changed from Windows to Linux since the Linux instances in the Hacklab have SSH running by default. This also allowed the worm to run using Python natively, at it is pre-installed on the Ubuntu installations running in the Hacklab.

2.2.2 Visual Payload

The payload was initially developed to dump a large number of text files onto the target's desktop. This was done using a simple loop to create a series of files with different filenames. During the fourth meeting *(Appendix 5.4)*, the clients pointed out that this was a clunky solution to the problem and something more simple like an image being displayed on the screen would be more effective. It was then decided that a full-screen image would be displayed on the infected machines as part of the infection process.

2.2.3 Self Propagation

Self propagation proved to be one of the more complicated aspects of developing the worm. The most simple part was looping through a list of hosts but for the actual processing of each host there were several individual processes that had to be implemented in order for the whole system to function properly.

Firstly, the worm had to check if the current host should be infected. Since the worm already had a valid list of hosts to loop through this acted as a whitelist. The worm then had to check if the target host was already infected. This meant that the worm itself would have to create a simple and subtle means of marking the occupied system as infected.

The solution to this found was a basic sockets server which would respond with "infected" when any data was sent to it. With this in place, the worm could then attempt to connect to the specified port, send some data, and check if the response was "infected". Any part of this process failing would indicate that the host was not infected and the worm would attempt to propagate.

Now that the worm had a means of recognising infected hosts it could loop through the potential targets, check if they are infected, and if not, attempt to infect them. As this loop runs forever - as long as the process hasn't been killed - if a worm is removed from one of the hosts on the list it will be quickly reinfected by one of the other worms on the network if any are present.

2.2.4 API Communication

As the worm checks for certain conditions it also posts the results of these checks to the API. For example, in the previous section it was described that a worm could check if a host was infected or not. In either case, the worm was made to post the current state of the target host to the server. This meant that the server would be continually being updated with the state of the whole network, making the real time visualisation more accurate.more accurate.

2.2.5 Anti-Worm

Removing the worm from the network was an extremely important part of developing the worm. Fortunately, it was not that complicated to implement since the worm itself already created a means of accessing the hosts and executing the commands (the SSH keys).

To remove the worm from a host essentially the processes have to be killed and the files created by the worm have to be removed. The worm starts several threads from the main thread, but within Linux these are represented as sub-process of the main "worm" process so just the main process has to be located and killed. However, the worm also starts an image viewer so that the blue screen image can be shown in full screen, so this also had to be searched for and removed. This was done with basic Linux process management tools over SSH.

Regarding the files created, there are two different reasons that files are present. First, there is the actual running of the worm which involves the worm file itself and second there is the blue screen image to be displayed. Then there are the SSH keys, used for access. Since the SSH keys are being used to execute these commands they must be removed last, but other than that the files can all just be removed normally using the SSH access.

2.3 ViMA

2.3.1 ReactJS

From the beginning of development Qubytes decided to use React to implement the front end of the web application. There were several reasons for this but the main reasons are that it's very optimised for dynamically changing information with asynchronous network requests and that the model of components and states makes developing sections with different information conceptually more in line with what actually needs to be implemented.

The app essentially needed to pull the information from the API and render it. React was ideal for implementing this since with React's states model the entire state of the network can be grabbed from the API, but only new or changed information will be changed in the web app's storage. This is a very efficient way to integrate with the API and it's exactly what was needed for how the API was designed.

The page was designed as several segmented sections showing different selections of information about the network; this also works with React's model. As the network state is stored in the parent "app" component, each section of the page was then implemented as a child component of the main app. This made it trivial to pass the necessary information down to each component, and as this information is part of the parent state it's only updated when necessary making every component and the app as a whole very efficient.

2.3.2 API Communication

For this project, Qubytes decided to use modern JavaScript methods to implement most functionality. For API communication, the API method had to be called with a network request, then the information had to be parsed and processed.

With the newest version of JavaScript - ES6 - executing network requests is made trivial by using the fetch() method. Using more ES6 methods, the data was then converted to a JSON object, which was parsed to update the state with and new pf changed information. This can be seen in the following snippet.

```
// get the JSON data from the API
fetch('http://127.0.0.1:8000/api/worms/?format=json')
    // convert the response to a JSON object
    .then(res => res.json())
    .then((data) => {
        // update the application's state with the new data
        this.setState({ hosts: data })
    })
```

In this snippet it can also be seen how these modern methods simplify the codebase and reduce the amount of lines quite significantly. To implement this using, for example, a XMLHttpRequest object would have been far more complex.

2.3.3 Components

Next, now that the application could get information about the network from the API, the components that would display this information were implemented. Each component was constructed with the application's state being passed into it by reference. Then the relevant information to be displayed was accessed through this state object. It was inexpensive to pass in the whole state since it is just a reference to the actual object in the parent component.

```
const Component = ({ info }) => {
    // processing of info happens here
};
```

And with access to the state of the whole network, any processing on that information can be done and any part of that information can be displayed. For example, to return an unordered list of all hosts as a JSX object, a simple ES6 map could be used:

```
<ul>
{info.hosts.map((host) => (<li>{host.ip}</li>)}
</ul>
```

The real elements are more complex than this as they use more HTML elements, but this demonstrates the concept behind most of them.

The next step was implementing a way to interactively "select" a host so that it's information would be displayed in the "Host Info" section as well as it's IP address being highlighted everywhere that it's displayed in the application. This "selection" had to be able to be done from any component, where if an address was clicked on, the host that it belongs to becomes selected.

This initially seemed like a difficult problem, but after looking into javascript objects more an ideal solution was found. The React state object was used to store a reference to a function in the root component that sets the currently selected host in the state:

```
this.childSelectHost = (value) => {
    // store the selected host in the state
    this.setState({ selectedHost: value });
};

this.state = {
    ...
    // reference to the host selection function
    selectHost: this.childSelectHost,
};
```

With this declared, the child components that have access to the state object could then set the currently selected host. This is useful as data in the state object can be changed from the root component, but not from the child components directly.

The most important place this was used was in the graph component.

2.3.4 Graph Component

For creating the graph the team deliberated over a few options. From a technological standpoint, either the graph could be implemented on a canvas object or as a collection of dynamically

updated SVG objects. Neither of these would be trivial to implement and both had their pros and cons, but since React has better integration with dynamically changing HTML objects the decision was made to go with the SVG method.

Since this would be difficult and time consuming to implement, the team decided to use an external library to create the graph component. Fortunately there are several open source projects for React based graphs, and after some research Qubytes decided to use react-tree-graph for ViMA's Graph. *(GitHub, 2020)b*

This presented a quite difficult problem of converting the data supplied by the API into a format that can be used by the react-tree-graph component. The problem is that the data supplied by the API was a flat array of worms that contained information about themselves and the IP of the parent that infected them. To be used by the graph component this had to be converted into an array with a tree structure representing the layout of the network, with worms being stored as children of their parent worm in the array.

Again the team took a modern JS approach to this problem by using chained ES6 methods. First, the root node's parent had to be specified which, in this case, is set as 0. Then each of the worms that had 0 as a parent - which should only be one - were created as a node in the tree array. One of the attributes of this node is the "children" prop which is used to recursively call this nest function with the IP address of the current host. The solution can be seen in the following snippet:

```
// recursively nest the flat array into a tree array
const nest = (items, id = "0", link = 'parent_ip') =>
    items
        // begin tree with root node
        .filter(item => item[link] === id)
        // for each item at this layer create a new node
        .map(item => ({
                name: item.ip,
                // recursively nest this node's children
                children: nest(items, item.ip),
                gProps: {
                        onClick: (event, node) => nodeClick(node)
                }
        }));
```

With this in place, the tree array could be passed to the component and the graph would be rendered properly. Another important attribute to note is gProps which is used by the react-tree-graph library to determine what happens when that specific node is clicked. The function called is nodeClick() which is just a wrapper around the selectHost() function as described in the previous section, the code of which can be seen here:

```
let nodeClick = function(node) {
        info.selectHost(node);
}
```

It can be seen that the function is accessed via the info object which contains the parent state. With the gProps of each node passing in that specific host's info to the function, the state will immediately be updated and all of the other components will also be updated with the new information.

2.3.5 Design

A draft design was made and added to throughout the building of the web application interface, used to test data fetching along with different tree graphs and researching CSS grids.

Mid-way through the development a final design concept was made using Draw.io and discussed. This design gave specific dimensions of each of the grid elements and the colour configuration of the text, background and tree graph. The final wireframe design is available in *(Appendix 6.6)*.

2.4 API

Early in the implementation phase of the project it was decided to use a REST API to enable communication between ViMA and the worm.

Different options were discussed between team members such as SQL/PHP. PHP was considered mostly because team members had experience in using it for past university work, but in the end Django was chosen with the Django REST Framework extension installed in our project. This is a powerful extension that made building web APIs in Django much simpler.

We decided on using this combo due to the interoperability of REST technology along with the benefits of using Django such as being able to run the API and web interface from a single instance. A final benefit is that it was based in Python which team members were also confident with.

During the development of the project, it became evident Django was more suited to our project due to the ease of implementing features such as REST and automatic URL routing. The use of Django in our project also offered Object-Relational Mapper (ORM) which abstracts databases to Python models making them easier to interact with.

2.4.1 Models

The ViMA worm model has 4 fields that declare which data is stored where when the application runs. Firstly a field to store the IP address of the machine posting the data, another to store the parent IP of the machine that infected it, one to declare it's current state of infection, and lastly one to store the time of last interaction with ViMA.

In Django, models were declared in a models.py file that other areas of the application would refer to as shown below.

```
class Worm(models.Model):
    ip = models.CharField(max_length=15,primary_key=True)
    parent_ip = models.CharField(max_length=30)
    state = models.IntegerField(default=0)
    last_activity = models.DateTimeField(auto_now=True)
```

2.4.2 URL Routing

Rather than the conventional method of URLs corresponding to file paths, Django allows for arbitrary parsing of URLs as a string using patterns. This means that a custom path can be used to activate various functionality within the application.

Django REST Framework enables extra functionality, in particular routers, viewsets and serializers. Routers are used to wire data to a set of URLs, viewsets are used to filter and order

this data, and serializers are used to format this data - in our case to JSON. The following example is how ViMA enabled querying and editing of the database via URL for machine IPs.

Qubytes used a serializer that declared how to format the request in a readable way.

```
class WormSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Worm
        fields = ['ip', 'parent_ip', 'state', 'last_activity']
```

After this, a viewset was created that declared the data to be fetched. ViMA's implementation fetches all the results, takes the formatting from the serializer, and restricts input to IPV4 addresses using regex.

```
class WormViewSet(viewsets.ModelViewSet):
    queryset = Worm.objects.all()
    serializer_class = WormSerializer
    lookup_value_regex = "(?:[0-9]{1,3}\.){3}[0-9]{1,3}"
```

Finally, a router was created with the viewset declared previously and the api/ path was chosen for accepting requests.

```
router = routers.DefaultRouter()
router.register(r'worms', wormapi_views.WormViewSet)
urlpatterns = [
    ...
    path('api/', include(router.urls)),
    ...
]
```

More advanced functionality can be implemented with wired URLs for example a /forgotpassword function for each user, but for ViMA this wasn't necessary as it was simply used to hold machine infection information.

2.4.3 Worm Communication

On the worm running on the machines, HTTP requests were used to communicate with the REST API. The following example is a GET request querying 10.0.0.27. This request was used to return a JSON containing all required information about the worm.

```
curl http://127.0.0.1:8000/api/worms/10.0.0.27/
{"ip":"10.0.0.27","parent_ip":"10.0.0.25","state":2,"last_activity":"2020-03-16T19:43:39"}
```

This made it easy to update ViMA from the worm as PATCH requests were used to update the server by just adding the infected host's own IP in the URL along with a JSON including the data to be amended.

```
    myjson = {'state': '1'}
    requests.patch("http://" + SERVER_IP + ":8000/api/worms/"
    + self.ip + "/?format=json", data = myjson)
```

3 Results

3.1 Overview of Results

This project ended up consisting of three main components: a worm, the ViMA web application and the API. All of these function together to provide a modular, easy to use and highly useful educational tool.

The worm successfully self propagates through the network while displaying it's infection in a very clear and recognisable way. It also continually updates the API with any new information it finds or any functions it carries out to alter the state of the network. It can be used independently of the visualiser to just show the spread of a worm using just the visual indicator, while the network indicator would easily allow a network administrator to monitor the presence of the worm just by scanning the network.

The ViMA web application is essentially an interactive, easy to use and engaging tool to show the spread of malware across a network in real time. With smooth animations as the worm spreads, and each component of the web interface updating in real time, it is very visually interesting to watch while displaying useful information all over. ViMA also has an easy to grasp interactivity which allows the user to display more information about each infected host, including which host it was infected by, time alive, and infection rate.

Finally the API that Qubytes has developed is a simple, generalised solution of retrieving and storing the state of a network that is being infected by a worm. It has been deliberately designed to be completely independent of the Qubytes worm and visualiser, and could be used by future developers - or even by Qubytes if the client requests - in order to create other visualisation systems or integrate with other worms. With a solution like this, there are many creative possibilities for usage that go beyond just what the team have implemented for this project.

3.2 Worm

The Qubytes worm is a harmless example of malware which can propagate through the Hacklab The worm carries out many functions simultaneously upon propagation. Firstly, it creates a visual indicator on the monitor of the target host by opening up an image of a blue screen, allowing a person looking at the hacklab computers to see the spread of the worm. This makes for a very visual demonstration of how the worm propagates through the network.

The worm also starts two processes necessary for propagation. One is a socket server that will respond with "infected" when any data is sent to it. This allows the worm to identify infected hosts in a simple way but could also be used to, for example, allow network administrators to identify which hosts are infected just by checking which ports are open. The other propagation process it the actual scan and infect process which loops over the white-listed hosts and checks if they aren't infected, and if they aren't, infects them.

The use of a whitelist ensures that no hosts outside of the intended targets get infected. Combined with the fact that propagation just uses harmless SSH methods, this worm is completely safe to use within the hacklab environment even when other people are using different machines on the network. This is useful from an education standpoint as it makes demonstrating the worm and the visualiser safe.

A simple to understand user interface is also built into the worm. All a user has to do is write out the whitelist of target hosts, then run "python3 worm.py infect [target host]" which will initiate the spread of the worm with the [target host] IP address. The worm has coloured output informing

the user exactly what is happening at each stage of initial propagation which is again highly beneficial for educational use. The user interface can be seen in figure 2.

```
jack@LAPTOP:~$ python3 worm.py infect 10.0.0.23
[10.0.0.23] propagating to 10.0.0.23
[10.0.0.23] Copying worm
[10.0.0.23] Copying private key
[10.0.0.23] Copying public key
[10.0.0.23] All files successfully transferred
[10.0.0.23] Executing on 10.0.0.23
[10.0.0.23] worm running on 10.0.0.23
jack@LAPTOP:~$
```

Figure 2: Command line view after executing the worm

As the worm's payload includes measures to ensure that no attempts are made to infect hosts which are already infected, the worm can work efficiently to spread through the network as effectively as malware would. Other features of the worm include a pause of five seconds between each stage of infection and propagation, so that the rate of infection is visible to people watching the tree graph on ViMA in real time.

3.3 Web Application

Using regular requests to the API, the Web Application is updated every second with the current state of the network and the progress that the worm has made. This information is then stored as a state of the root component of the application. From this, each child component of the application can be immediately updated with the new relevant information it needs without many different requests, optimising and minimising network traffic.

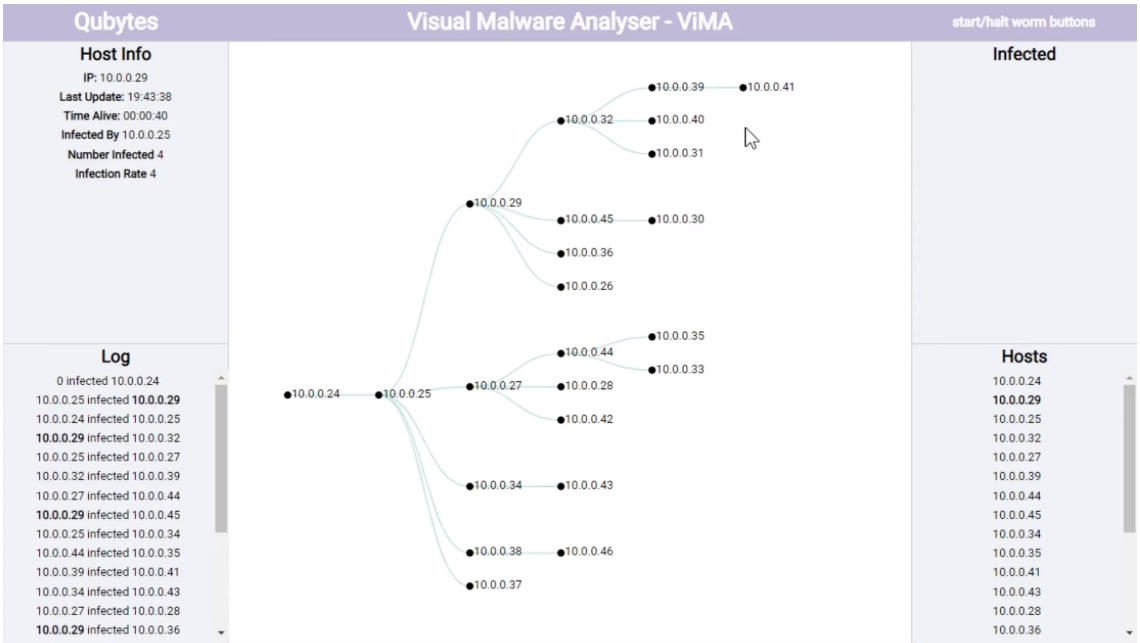


Figure 3: View of web interface during live infection

The web application, seen in figure 3 is split into five main blocks, which convey different information: infected and discovered host lists, the infection tree diagram, an infection log and information on specific hosts. As each of these are implemented as components it was simple to implement and modify components as the clients requested, while also making the web interface well optimised from a performance standpoint.

The main section of the web application contains a visual representation of the worm's spread and infection throughout the network, in the form of a tree graph. This is the most important element for two reasons: it's the centrepiece of the interface and it's the main feature requested in the brief. As the worm spreads throughout the network, the graph updates in real time with smooth animations to show the current state of the network. This format easily conveys the infection paths taken by the worm and it successfully visualises the rate of propagation in an engaging way.

Interactivity is introduced with the ability to click on nodes of the graph and get specific information about that node. This also highlights the IP of that selected host wherever it shows in the web application. This makes the application more engaging especially once the propagation has finished and the user would want to review what happened in an interesting way.

3.4 Back End

The API is an interface to the database on the server, allowing clients to update the server with information about the network being attacked and request information about the current state of the network. Since these apps should be used within a Django project, the user could configure what type of database they would want to use on the server among other things configurable from a Django project.

ViMA itself is also running on the back end, but mostly just to serve files. Most of the app's functionality is compiled into a single JavaScript file which is then served up to any user who accesses the web site. Once the app is running on the client, all of the processing is done from there including API requests to get the data and parsing of the data received.

So the two components of the project that run server-side - the ViMA web application and the API - are both implemented as separate Django applications. One benefit of this modular approach is that in the future developers could use the ViMA API to develop their own visualiser systems, or they could develop their own API using a different technology, then use the visualiser app to interact with it.

4 Discussion

4.1 General Discussion

As the previous sections explain, ViMA meets all the requirements set out in the client brief. The group, Qubytes believes that it is a valuable learning resource that can be used to effectively visually represent the spread of a worm throughout a network. Detailed below is how the product produced meets all of the requirements set.

The worm fills the requirement for the worm to be harmless, as it spreads using the legitimate method of SSH, which is standard traffic between computers, as shown in the Worm section of Procedure. This is instead of using am exploit, such as EternalBlue, which can compromise system security and stability.

ViMA monitors the system and network information, as it has an API that the worm communicates with to update the web interface. Satisfying a key aim of the project; to create a clear visual display of worm propagation. This is as when a machine is infected it will send a response the central server every 5 seconds to update it's status (discovered/disinfected or infected), along with its current IP address and the IPs of machines it then has infected.

The information about the propagation through the network is shown using the group's React web interface. The interface gives a timeline showing which machines have infecting which, along with a list of the IP addresses both discovered and infected. The main focus of this project, however, is the tree graph which gives a live view of the worm spreading and branching out from the original node. The interface also visualises the actual infection rate with a counter of how many machines a single machine has infected.

Some changes were made throughout the development of ViMA, as to be expected of a project of this type. Due to the groups adoption of the prototyping methodology this was accounted for and so were discussed with the clients regularly to make sure the final product would suit them.

Although the project was designed to be as universal as possible, due to the specific setup in the hacklab some changes in commands and syntax would be required in order for the worm to run in different UNIX environments and configurations. Major changes would be required to run the worm and enable propagation on a Windows system due to significant differences in how SSH and machine communication work on the Operating System.

4.2 Conclusions

The final product exceeds all requirements set out by the client in the initial briefing.

ViMA is a tool which will be able to be aide in teaching and learning about malware propagation. It visualises the spread of malware throughout a network in an easy-to-use, easy-to-understand way. The Qubytes worm is effective in clearly showing the infection of each host without infecting any machines with harmful malware. The anti-worm can stop all of the worm's processes and remove all created files on each host just as quickly and easily as they became infected.

The use of visualisation throughout the product - particularly in the worms propagation, the infection tree diagram, and the sectioned layout of the visualiser application - was implemented to aide in teaching about malware as the very clear level of visualisation aims to provide a deeper understanding to students, than that of text or static diagrams.

The modular structure of ViMA resulted in the the product being very flexible. The use of an API with a database, in which stores information on the worm's infection, to populate the web

application with information means that it is very easy to use 'fake data' to populate the database. Which can be used with the web application to simulate an infection.

The API can also easily be used with a different visualiser, if it was required. The separation of the different components allows for very easy alteration of the product, and isolated used of different parts. This means that the team may easily add to/improve the product in their future work, and that people can make use of the specific aspect of the product that they need.

Choosing Django to run the visualiser and the API, as opposed to another framework, provided numerous benefits to both the group and the project overall. Since Django is capable of running multiple applications from instance it allowed the group to run both ViMA and the API. As mentioned previously, in the "Procedure" section, the functionality of this framework allowed for highly automated configuration and features like automatic URL routing.

The JavaScript React framework used within the implementation of the API is designed for dynamically updating content and allows for ViMA to update in real time. This is a crucial part of the display as it would not be very user-friendly if the end user had to refresh the page continuously in order to see the network traversal of the worm. The react-tree-graph library also displays the information in a clear and concise manner, by implementing an pre-existing library rather than creating one from scratch Qubytes was able to keep on schedule and more support was available online.

The prototyping methodology chosen by the team ultimately proved very successful in ensuring that the product developed met it's optimum potential. The regular presentation of prototypes to the client helped the team refine the product in the client's vision. The feedback provided by the client gave the team insight into the their needs that the team wouldn't have otherwise had, had we left all communication with the client until the product was completed or nearing completion. Some key feedback from the client, which shaped the team's implementation of the project included focusing on a disinfection functionality in the worm, and interactivity with the tree graph on the web application.

Throughout the entire development stage, every team member tried their hand with at least two technologies/languages that they had little to no experience in. All members of Qubytes have expanded their knowledge and skill set whilst working on the development of ViMA. This includes a deeper understanding of network protocols and the use of API's, a higher skill level of python scripting and JSON development, as well as increased knowledge in malware development, and client communication and professionalism skills.

4.3 Future Work

Unfortunately during the later stages of this project there was an unexpected global pandemic that lead to the inaccessibility of the universities facilities, which were being used as the testing and development environment throughout this project. This resulted in the current implementation of ViMA and the Qubytes worm having some limitations that, given more time and resources the team would aim to negate or improve upon.

Currently, there is no way to deploy the worm to the network via ViMA. The worm must be deployed from a computer within the target network, however from that point it is completely self propagating. Allowing the worm to be deployed from the computer running the ViMA software introduces another level of complexity into the worm, as it would require a specific payload different from the usual worm payload, which runs only when the deploy button is pressed. This individual payload would have to not run the blue screen visual. It would then have to infect only one host on the target network, then stop infecting, so that there is a single starting point from within the target network which can be displayed as the parent node in the tree diagram. This

payload would also have to send a specific host name to the API to distinguish itself from the target network, as to not show up as an infected host in the visualiser application.

There is currently some visualisation of dis-infection shown on ViMA as the nematode works it's way through the network but this is limited in comparison to the level of visualisation of the worm's path through the target network. With more time, the team could add a infected/disinfected colour aspect to the tree graph to show the disinfection of infected hosts, as shown in the original web page wire frame.

The worm currently only runs on a specific version of one operating system; Ubuntu Linux (16.10). Given more time to work on exploring different means of propagation and infection, the original project could be improved in order to include a non version specific Linux based worm and a Windows based worm.

Finally, ideally there would be a way to record and play back data that was fed to the API, potentially via some sort of time control mechanism on the ViMA interface. This would mean that instead of having to feed data to the API to use the animations and see the propagation happening, a user could just use some pre-existing data to demonstrate the propagation. It could even come with preset data sets that could be used to demonstrate different types of propagation.

4.4 Call to Action

Although this project was initially designed and created as a free, open-source product future improvements, such as the ones mentioned prior, as well as continued technical support could be made as part of a paid subscription or one time purchase. Although due to this project's nature and the relationship of Qubytes to the clients a reduced rate or free lifetime subscription would be considered.

Due to the very nature of ViMA's built in ease-of-use and amount of clear documentation, both in clear code comments and this white paper, the group does not feel it necessary to have to offer training on this product. Although if there are any further queries on either how to use ViMA or on future work the group can be contacted through their communications manager, **Mairi McQueer: 1700231@uad.ac.uk**

The link to the project in its current state can be found within the references under *GitHub, 2020a.*

5 References

Center for Internet Security (2019). *Top 10 Malware March 2019* [online]. Available at: https://www.cisecurity.org/blog/top-10-malware-march-2019/ (Accessed 25 March)

Check Point Research (2017). *EternalBlue – Everything There Is To Know* [online]. Available at: https://research.checkpoint.com/2017/eternalblue-everything-know/ (Accessed 17 January)

Django REST framework (2020). *Django REST Framework* [online]. Available at: https://www.django-rest-framework.org/ (Accessed 13 February)

GitHub (2020)a. *malware-visualiser* [online]. Available at: https://github.com/Samiser/malware-visualiser (Accessed 28 January)

GitHub (2020)b. *react-tree-graph* [online]. Available at: https://github.com/jpb12/react-tree-graph (Accessed 06 March)

Python (2020). *Python* [online]. Available at: https://www.python.org/ (Accessed 17 January)

The Telegraph (2017). *NHS Cyber Attack: Everything You Need To Know* [online]. Available at: https://www.telegraph.co.uk/news/2017/05/13/nhs-cyber-attack-everything-need-know-biggest-ransomware-offensive/ (Accessed 6 April)

The Virus Encyclopedia (2012). *Nematode* [online]. Available at: http://virus.wikidot.com/nematode (Accessed 25 March)

Acronyms Used

Visual Malware Analyser (ViMA)
Center for Internet Security (CIS)
Operating System (OS)
Secure Shell (SSH)
Object-Relational Mapper (ORM)
JavaScript (JS)

6 Appendices

6.1 Worm Logic Flow

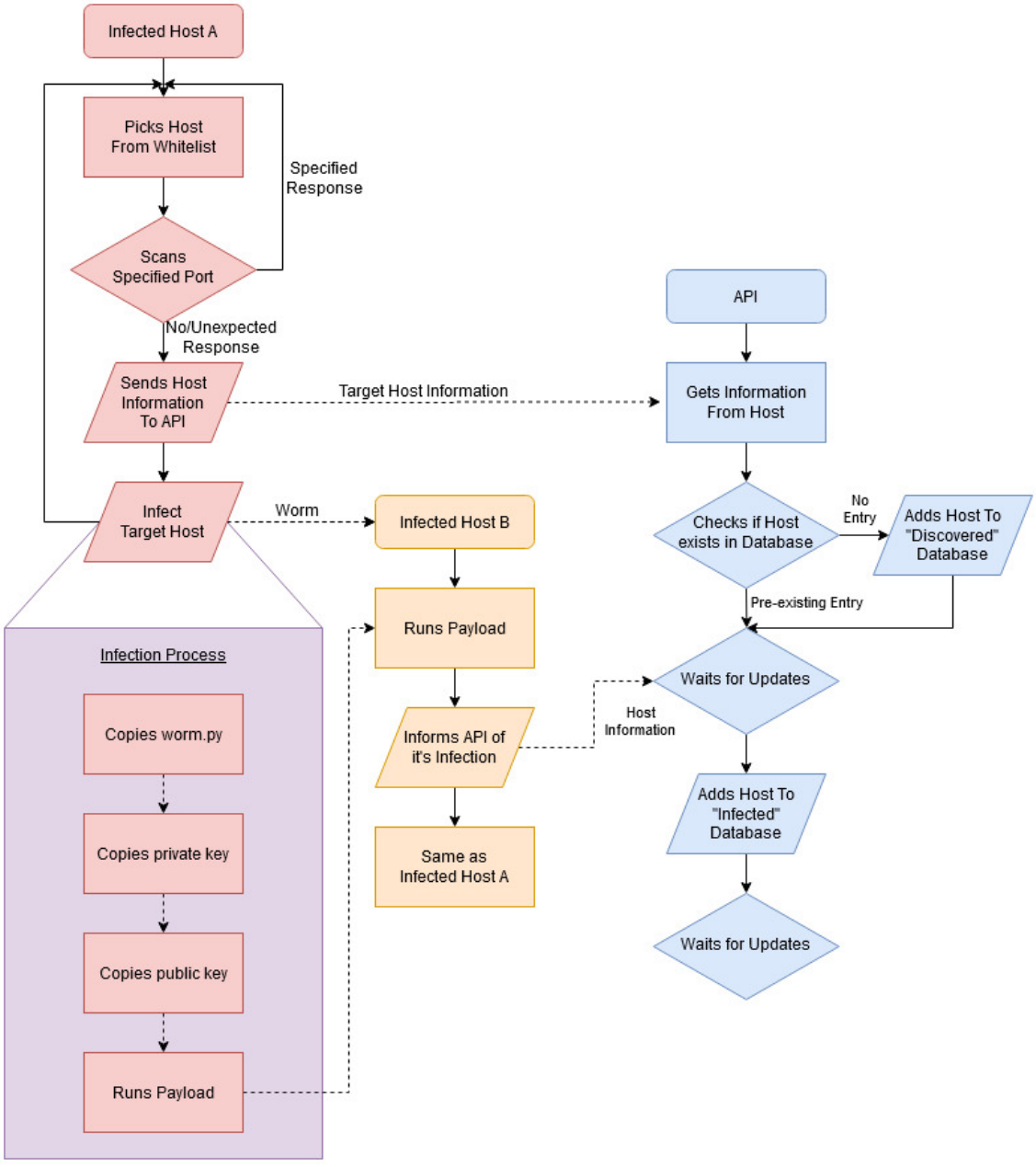
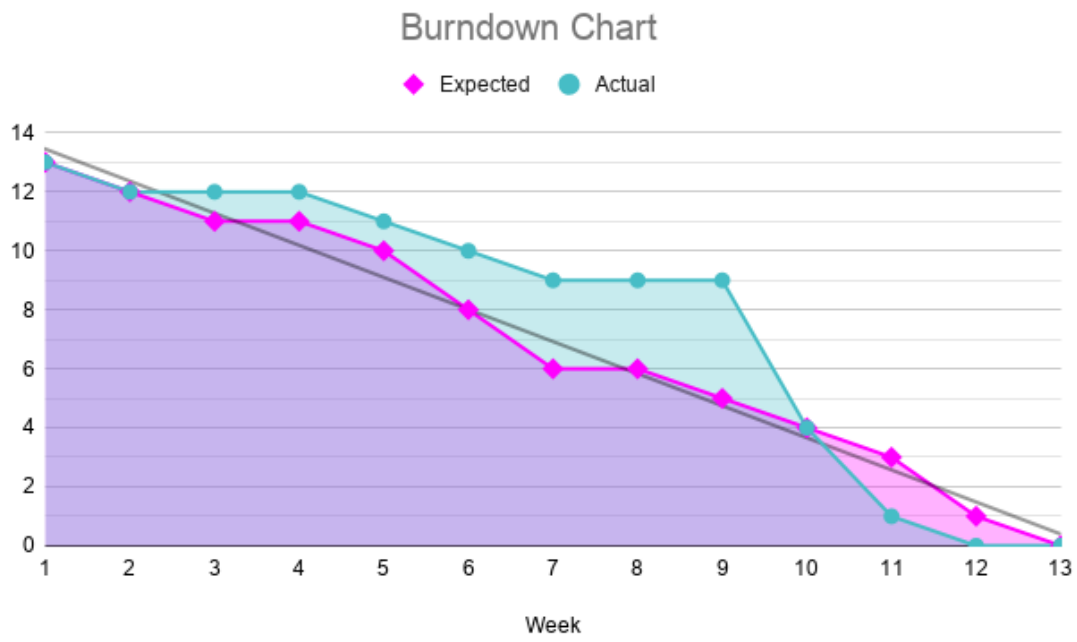


Figure 4: Flowchart Detailing The Process of The Worm

6.2 Project Burndown Chart

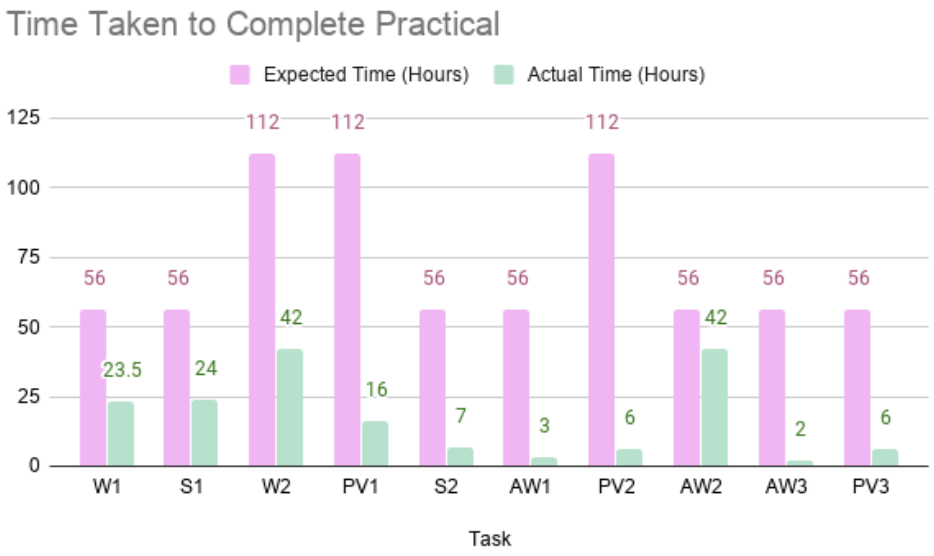


6.3 Comparison of Time Taken to Complete Tasks

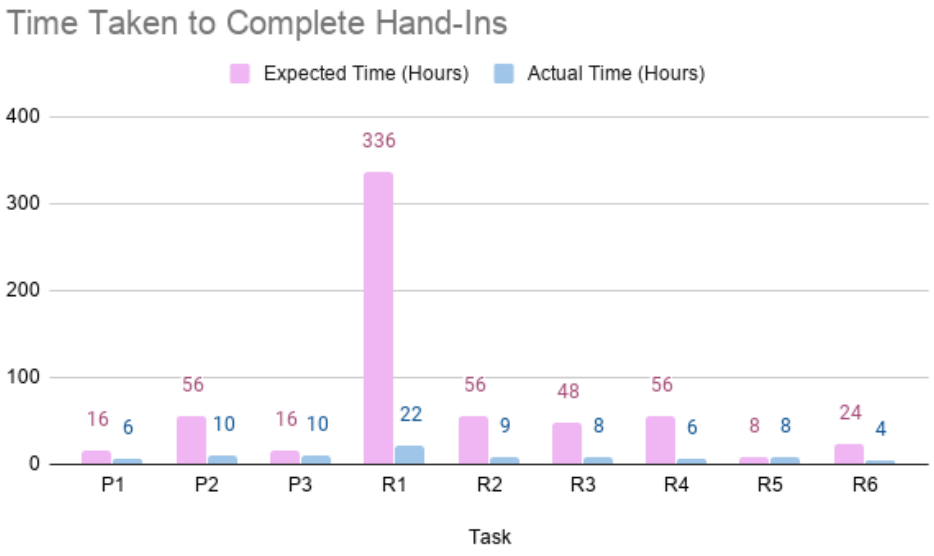
6.3.1 Key for Tasks

Design and create worm delivery method	W1	Make first draft of presentation	P1
Server and worm communication	S1	Review and edit presentation	P2
Implement worm propagation	W2	Recording and editing of presentation	P3
Design and create web page front end	PV1	Procedure and methodology	R1
Sending data to database	S2	Introduction	R2
Implement method of discovering infection	AW1	Results	R3
Design and create web page back end	PV2	Discussion	R4
Implement anti-worm propagation	AW2	Abstract	R5
Implement 'killing' of original worm	AW3	Referencing/Appendices	R6
Implement web page updating in real time	PV3		

6.3.2 Practical

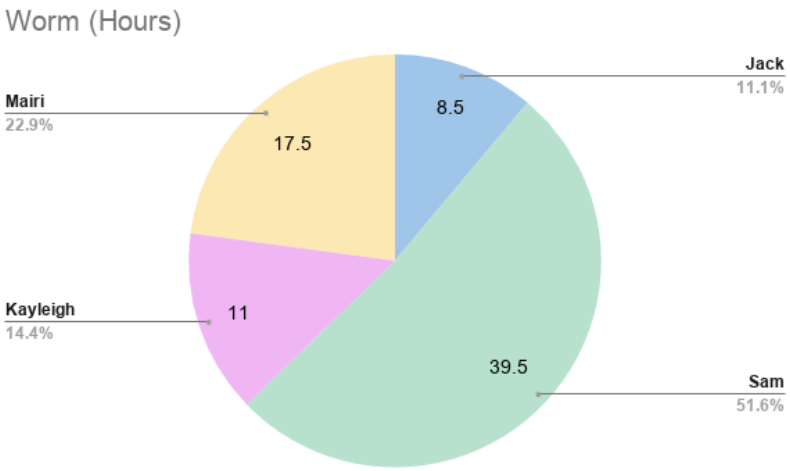


6.3.3 Report and Client Pitch

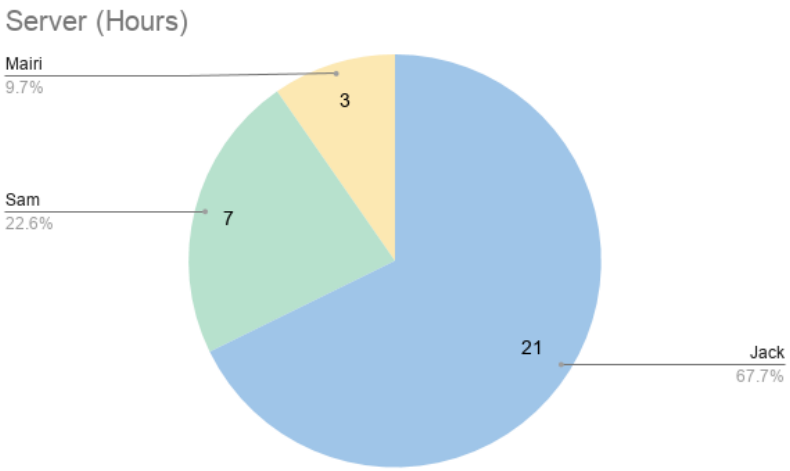


6.4 Team Contributions

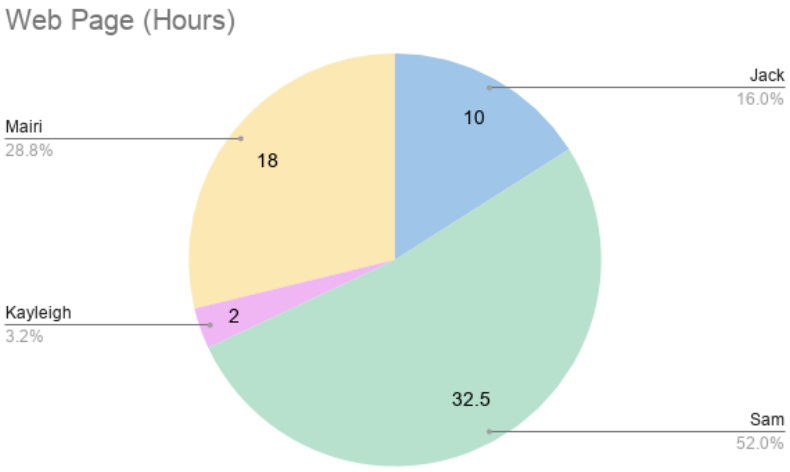
6.4.1 Worm



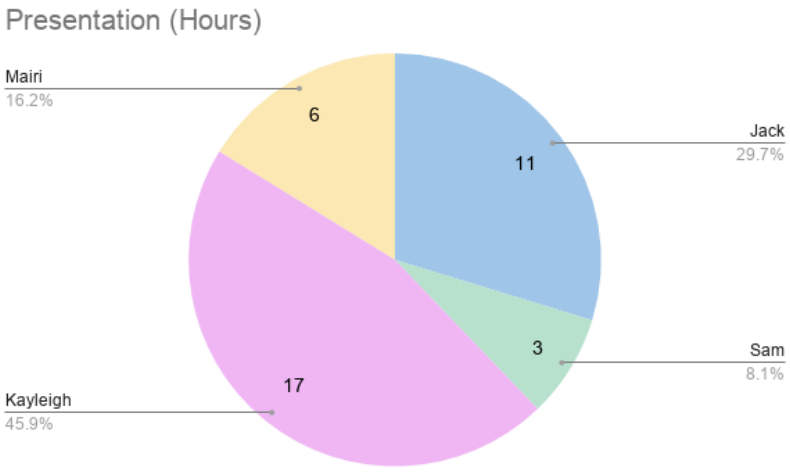
6.4.2 Server



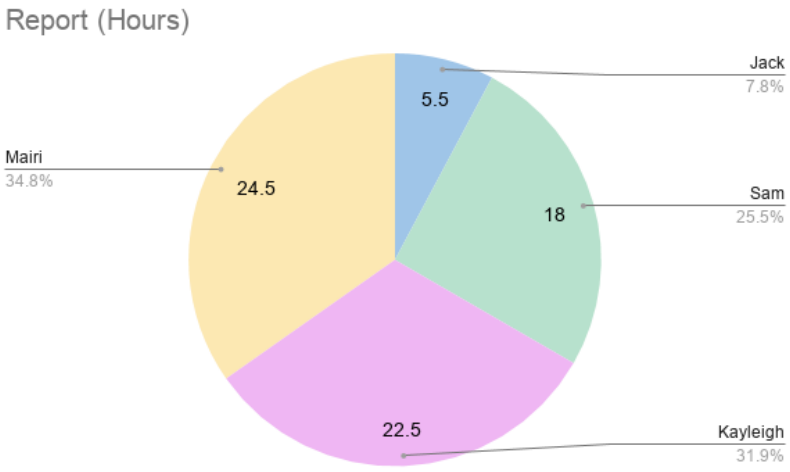
6.4.3 Web Page



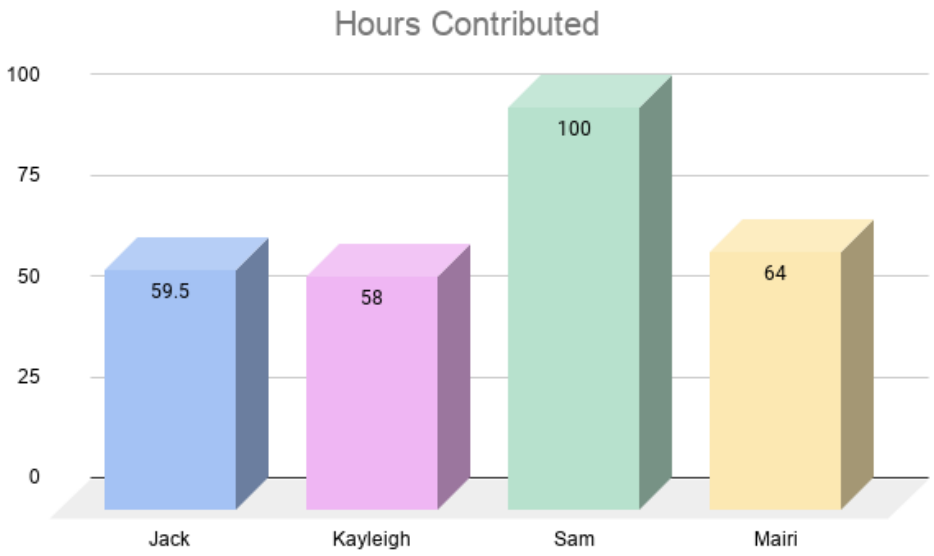
6.4.4 Presentation



6.4.5 Report



6.4.6 Total Hours Contributed



6.5 Meeting Minutes

6.5.1 Meeting One (17/01/2020)

Apologies: Kayleigh (Could not attend due to illness)
Present: Jack, Mairi, Sam
11:30am

- Deliverables and Requirements' forms were printed out and filled in by Mairi.
- All team members present discussed the project in order to refresh everyone and to understand where each person was starting.

12pm

- A regular meeting time was discussed and agreed upon; 11am Fridays.
- How team contributions will be monitored was discussed, Sam and Mairi had a disagreement over this. In accordance with how the group planned to solve these debates, a vote was held and it was decided that members would detail their contributions on a shared spreadsheet that would then be used to create weekly burndown charts.

1:10pm

- Requirements were discussed with both clients. The group was asked about how they plan to control source code, allowing group members to all access it without making it publicly available. The clients also inquired about how the removal of the 'harmless' worm would be ensured.
- Once the clients were satisfied with the project requirements and deliverables they signed and dated both forms, along with the present group members.
- A weekly meeting time of 1 pm Fridays was agreed upon by the clients and Qubytes.

2pm

- Meeting concluded, Sam and Jack started looking into exploit development and worm propagation.

6.5.2 Meeting Two (31/01/2020)

Apologies: Kayleigh (Could not attend due to illness) and Sam (Was late to the meeting)
Present: Jack, Mairi and Sam
2:45pm

- Meeting with client
- Discussed issues and proposed changes:
  - Eternal blue on windows using smb with Python but eternal blue with not exploit without using meterpreter or having access to smb pipes
  - Proposed solution: Using SSH in Linux instead and not an exploit, copying over a file and opening it remotely
  - Clients agreed with changes and proposed that if the group has time that they revisit the eternal blue exploit
- Discussed the Django REST API that is currently being implemented for communication between the worm and the server (ViMA)
- Client suggested we ensure all changes and reasoning's behind such are documented clearly for ease of write-up in the whitepaper

6.5.3 Meeting Three (13/02/2020)

Apologies: None
Present: Jack, Kayleigh, Mairi, Sam
2:45pm

- Sam demonstrated to the rest of the team the progress that he had made on the worm propagation and payload
  - Carrying SSH authentication keys
  - Socket with 'infected' reply
  - Disinfection process
- The team also reviewed Sam's progress on the visualiser web application
  - Requests to the API
  - Use of django

6.5.4 Meeting Four (18/02/2020)

Apologies: None
Present: Jack, Kayleigh, Mairi, Sam
1:40pm

- The main visualisation method for the infection on the target network was discussed. A tree graph was decided on within the group
- Kayleigh sourced the "d3" library with open source graphs, and this was looked into, but ultimately decided against when none of the graphs available fit the style that the team was looking for.
- Mairi then sourced the Go.js library, which was reviewed and discussed by the team. It was decided to pursue this library further.
- Tasks for the team members were assigned to be worked on for the next week
  - It was decided that Mairi will look into the Go.js library, and try to gain understanding of the REACT framework to add to the web application
  - It was decided that Jack will work on a script that would populate the API's database with fake host data for web application testing purposes
  - It was decided that Kayleigh will work on converting the existing worm code into an object-orientated format
  - It was decided that Sam would take a break from contributing to the workload this week, due to the large amount of work he had contributed the week before

6.5.5 Meeting Five (21/02/2020)

Apologies: Jack and Kayleigh (Could not attend due to other work commitments)
Present: Sam and Mairi
2:30pm

- Showed clients worm infecting a computer in hacklab
- Discussed worm progress and issues had with implementation
  - Used subprocessing to execute bash commands with Python3
  - SSH requires password so team had to implement a method of initially entering a password to then execute SCP to transfer the keys so then can use them
  - Use port 6969 to communicate between worm and server

2:45pm

- Discussed visualiser progress
  - It was realised that Go.js was not in budget (£4k license) so it was decided Mairi was to return to looking into the d3 library instead
  - Using REACT to update webpage in real time
  - need to work on updating when worm is removed and displaying accurate data

2:50pm

- Clients asked about maybe making worm infection more subtle, not having 200+ 'You've been hacked.wrm' files, maybe changing wallpaper colour or having notifications
- Clients reiterated that the IP range for the hacklab should be hardcoded

3pm

- Discussed groups use of REACT and JSX within the project
  - Clients seemed impressed with Qubytes progress

6.5.6 Meeting Six (02/03/2020)

Absent: None
Present: Jack, Kayleigh, Mairi and Sam
12:10pm

- The progress made by individual team members over the past week, as per the tasks assigned last week, was discussed.
  - Kayleigh has split up the current worm code into different classes.
  - Mairi continued doing research into the d3 library for front end graphics.
  - Jack has been working on making the worm do requesty stuff.
  - Sam has taken a week to provide help to the rest of the team, due to completing his work early.

12:30pm

- Tasks for the team members were assigned to be worked on for the next week.
  - Mairi will further research the REACT library to figure out how to implement that into the project.
  - Jack will try to add parent IP functionality into the worm code
  - Kayleigh will work on implementing the new object oriented code in main.
  - Sam will expand the api methods to update the worm status. And implement a skeleton of the web interface.

12:40pm

- The next meeting had been scheduled for 12pm the following Monday.

6.5.7 Meeting Seven (06/03/2020)

Apologies: Kayleigh (Could not attend due to other work commitments)
Present: Jack, Mairi, Sam
1:50pm

- Group discussed progress
  - Creating worm variables
- Decided Jack and Mairi should swap roles, creating the web page front end and implementing worm propagation respectively
- Mairi decided to assist Kayleigh with putting the existing python code into classes

2:15pm

- Jack and Sam started looking into the react-tree-graph library and integrating it with the web page

2:45pm

- Client meeting
  - Showed the client our progress on web page
    - Real time updating of infected machines
    - Explained what each part of the web page does
    - Tree graph doesn't render on a canvas so may be dynamically resized
    - Client asked if branches will be able to be expanded/closed, group explained that that is a goal
    - Using CSS grid to layout elements
  - Explained what group is attempting with the worm propagation python code
    - Using sockets to go from one machine to the next
    - Worm sends information back to server

3:00pm

- Jack and Mairi had a disagreement on background aesthetics for the web page, as team leader Sam had the final say and the background was decided to be white

6.5.8 Meeting Eight (08/03/2020)

Apologies: None
Present: Jack, Kayleigh, Mairi, Sam
1pm

- The team discussed how everyone was getting on with their individual tasks.
- Mairi explained that she was struggling with the react code. So Jack said he could take over the React code, as he is more confident with JavaScript.
- Kayleigh was confused about Python syntax when converting the current worm code into object orientated code. Sam offered to help out.
- Kayleigh will work on drafting a presentation for the client pitch.
- Sam will expand on the basic webpage he created with react, by linking it with the API/databases and integrating Jack's graph component

6.5.9 Meeting Nine (13/03/2020)

Apologies: Mairi (Could not attend due to prior arrangements), Sam (Late to attend meeting)
Present: Jack, Kayleigh, Sam
1pm

- The team presented the current implementation of the worm and malware visualiser to the Clients
- The clients presented their feedback including areas to improve and other advice
  - The clients require for an implementation of the visualisation of the 'reaper' - worm disinfectant.
  - Zoom and scroll functionality of the tree graph is desirable, but not as essential as a visualisation of the path of the reaper worm.
  - Be wary of taking on any large tasks as the product deadline is nearing. The team should define a strict end date and ensure their product is functioning well by that date. Anywhere that the final product is lacking at this point can be addressed in the 'future work' section of the product report.
  - Be prepared for the large workload which may arrive after testing the worm + visualiser, as the worm will propagate exponentially and the visualiser application must be able to keep up with this.

6.6 ViMA Design