The University of Manchester

School of Computer Science

Project Work (2013/2014)

Progress Report for

Whitelisting software

Daniel Dresner

Xue Fu

Contents

Abstract

The rapid growth of malicious code is a serious issue that threats enterprise endpoints and personal computers. However, traditional antivirus software cannot protect computers efficiently, threats such as zero-day attack are able to pass through antivirus and inject computers. Recently, whitelisting approach has gained momentum. In this paper, a whitelisting software is proposed and described in detail. It combines with behavioural and blacklisting approach in order to address shortcomings of each approach.

1 Introduction

The emergence and development of computer and communication technologies have radically changed the way that people communicate and exchange information with others. However, threats related to computer security also appear, such as worm, virus, Trojan House and spam. Methods and technologies for protecting endpoints from these threats have gained evolving attention. A widely applied method nowadays is blacklisting principle, which is implemented by a majority of antivirus software. Besides it, behaviour-based and whitelisting technologies are other common ways.

1.1 Approaches of computer security

Blacklisting is a traditional method to prevent known malicious code from executing, it is also the core technique applied for filtering spam. In terms of computer security, when a malicious code is identified, the digital signature of this code will be recorded on black lists. Applications whose signatures are on black lists will be detected and then quarantined or deleted. Although blacklisting approach predominates among present antivirus software, it has shortcomings and faces the following security threats:

1) Zero-day threats. Antivirus vendors need to spend a period of time to discover the latest malware and then update their black lists. During the blank time, the latest malware may attack or infect people's computers successfully (Pareek, et al., 2012).

2) Black list has to cover all signatures of identified malware, whose quantity is extremely large. According to Kim, et al. (2010), when a new malicious code is identified, the signature has to be added to black lists. With the increasing number of new threats appearing, the maximum capacity of black list is nearly reached.

3) End users may not update their black list on time. Thus, their computers are more likely to be infected.

4) Rogue security software is also a security threat to end users. It reports alarm of virus or spyware which actually do not exist on users' computers, and recommend user to purchase the full version (which probably provide limited or none function or even contains malicious code) to clean their computers[1]. However, blacklisting prefers to focus on preventing malware not these scareware.

Behaviour-based technology aims at identifying malicious code based on its characteristics and patterns. The methods of behaviour analysis include two ways. One is analysis of binary code to discover suspicious behaviours. Another is executing applications in a specific environment and monitoring their behaviours (Kim, et al., 2010). This approach can be an efficient way for recognising new threats. One problem is this approach is time consuming.

Approach applied by whitelisting is opposite to that of blacklisting. As mentioned before, blacklisting aims at exhaustively searching for all existing malware and protecting computers from those attacks. While whitelisting focuses on identifying wanted applications and only permitting execution of these applications. In other words, whitelisting applies the concept of "default deny" instead of blacklisting's "default allow" (Huh, et al., 2009). Whitelisting has many pros:

1) Whitelisting can be an effective way to defend zero-day attacks since new malware or spyware are not on the lists[2].

2) The number of applications' digital signatures stored on white lists is significantly lower than that on black lists. As Harry Sverdlove, the chief technology officer of Bit9, said: "What you want running on your system is a much smaller set than what you don't want."[3]

3) Whitelisting applications do not need to update frequently. This is because all files which are not on the list will be refused not matter when they are released.

---

[1] Rogue malware infections – what you need to know http://www.bullguard.com/bullguard-security-center/pc-security/computer-threats/rogue-malware.aspx

[2] The pros and cons of behavioural based, signature based and whitelist base security http://www.windowsecurity.com/articles-tutorials/misc_network_security/Pros-Cons-Behavioral-Signature-Whitelist-Security.html

[3] Whitelisting Vs Blacklisting http://kevtownsend.wordpress.com/2011/08/24/whitelisting-vs-blacklisting/

4) For enterprises, governments and other workplaces, time wasting applications, such as computer games, can be rejected to run by using whitelisting.

In summary, the primary advantage of whitelisting approach is applications which are not on a white list cannot be executed. However, this characteristic can also be a drawback as legitimate software not on the list cannot run likewise. Another shortcoming is the limitation of applying whitelisting. As the whitelisting approach only allows specific files to go through, the configuration of the list should be: (1) fully considered, in order to include all wanted applications; (2) conducted by computer security experts, in order to ensure applications on the list are not malware. However, current whitelisting software either uses one list for all customers or service for companies with security experts (Gates, et al., 2012).

1.2 Explanation of the project

In the first section, the characteristics of common security approaches are described, some advantages and disadvantages are discussed. In this section, I will briefly introduce my project, whitelisting software.

Whitelisting approach gains momentum recently, but as mentioned before, it has weaknesses and maybe not practical to some extent. For the project of whitelisting software, I decided to combine the three approaches together, in order to address previously mentioned issues of each approach, and make the software multi-functional. Basic functions of the whitelisting software are introduced below, which followed by a description of how it solves the issues.

The main function of the whitelisting software is the prevention of any executable files from running if they are excluded from the white list. Apart from that, it provides additional functions: unknown applications' behavioural detection and blacklisting files' refusal. To be specific, the software contains two lists, one is white list and the other is a black list. When an unknown application (which means it is on neither the white list nor the black list) attempts to run, it will optionally be checked in terms of their patterns and characteristics. Here, "Optionally" means users with the privilege (users have the right to manage the white list, such as administrators) have multiple choices:

1) Running the unknown file directly. Thus, behavioural detection will not be performed. The signature of the file will be recorded on the white list.

2) Stopping running the file.

3) Implementing behavioural based detection. Then, the behaviours of the file will be presented, for example, what system files it accesses and modifies.

If users chose the third option, they need a further judgement of executing it or not based on the presentation of its behaviours. If yes, the signature will be stored on the white list; otherwise it will be stored on the black list. The role of the black list is to record these judged files to avoid repeating behavioural detection.

The main contribution of the project is it synthesizes the three common security techniques and addresses main issues of each one. It will be described in detail:

1) In terms of the primary trouble of blacklisting, zero-day attacks and large-size malware database, whitelisting is implemented to solve them.

2) Regarding the limitation of whitelisting, the behaviour - based approach is applied. It provides a characteristic description of unknown files to assist users in identifying trustworthiness of files. Thus, business or governments may not need security experts to configure white list.

3) For behavioural technique which is time consuming, the two lists can largely reduce the frequency of executing behavioural detection.

Due to the features and drawbacks of the three approaches, I think it is a reasonable contribution.

1.3 Paper structure

The rest of the paper is organised as follows. In the second section, I describe the general background research which includes current whitelisting products and related papers. This is followed by specific techniques research which is closely related to my project, such as hash functions, hooks and signature. The fourth part presents the whitelisting software in detail and shows the progress of my project. In the last chapter a project plan is shown, which interprets what is going to be performed next.

2 General background research

In this section, general background research which refers to both whitelisting products produced by commercial companies and academic papers contributing to improving whitelisting and behavioural techniques is introduced. Specifically, commercial security software which implements the whitelisting approach to protect computers is illustrated in terms of their functions and

features. It is followed by a description of academic research in the field of whitelisting and behaviour-based detection.

2.1 Bit9 Security Platform

Bit9 is a computer security company which concentrates on the implementation of whitelisting. It broke the dilemma of the traditional security measures which cannot prevent present advanced threats, zero-day threats and targeted malware attacks. What's more, Bit9 does not merely applies the whitelisting approach, but it improves this approach by additionally implementing some useful techniques such as "detonate-and-deny" approach, files' trust rating mechanism and real-time recorder. This specific application whitelisting created by Bit9 is named "Bit9 Security Platform".

Bit9 Security Platform applies three core technologies in order to provide five primary functions to protect enterprises' endpoints against various malicious threats[4]. The core technologies are (1) real-time sensor and recorder, (2) real-time enforcement engine, and (3) software reputation service. The five functions it provides are visibility, detection, response, prevention and network security integration.

Real-time sensor and recorder is deployed on every endpoint and server in a company, it monitors and records all behaviours of files on every facility, which can be used for (1) real-time visibility, security administrations can gain knowledge of what files is running on which machines, what the files did; (2) further analysis, for example, security staff can identify suspicious executable codes based on the records of files' history behaviours.

The real-time enforcement engine provides two ways for protecting servers and endpoints. The first one is "detonate-and-deny" approach[5]. For every new executable file which attempts to run, Bit9 will transfer it automatically to the third parties FireEye or Palo Alto Networks to check the legitimacy of the file. If it is recognised as malicious code, Bit9 will reject the running attempt. The other one is called "default-deny". This means enterprises decided their trust policies, for instance, software from some specific trusted vendors can be executed on endpoints. Only files which meet the trust policies are permitted to run.

Software reputation service is a cloud-based service. It searches existing applications, calculates hash values, collects applications information such as

[4] Bit9 Security Platform https://www.bit9.com/solutions/security-platform/#overview
[5] Advanced Threat Prevention https://www.bit9.com/solutions/protection/

vendors and popularities, and provides trust ratings of applications. The service “contains billions of records and is the world’s most reliable source of software trust”[6]. It can be used to configure trust policies to protect enterprises’ endpoints and servers.

Bit9 Security Platform enables security administrators to have knowledge of actions, locations and behaviours of applications on their machines. Apart from that, enterprises can configure their own white lists by defining trust policies or by having new applications analysed by FireEye or Palo Alto Networks.

2.2 Faronics Anti-Executable

Faronics Anti-Executable is a threat protection product which applies the approach of whitelisting. In terms of enterprise level, it enables administrators to manage and configure endpoints easily from a single computer by utilising Faronics Core. Faronics Core is a central management tool for controlling endpoints, which also is implemented by Faronics other products, such as Deep Freeze. By providing the following features, Anti-Executable is probably an effective security software for enterprises.

1) By scanning a hard drive, it can build a white list which contains all wanted executable files, other files which are neither wanted nor authorised will be prevented from running. Besides, it provides other methods to configure and update the list, such as defining trusted publishers, specific applications and applications in a specific folder.

2) Anti-executable enables automatic updates of the list. When software from a trusted publisher updates, the white list will update automatically.

3) Like Bit9, Faronics provides an online database named IdentiFile containing millions of software’s hash values[7]. Customers can scan the database to identify the trustworthiness of a new executable file.

4) Another advantage is the central management of all endpoints within an organization. As mentioned before, administrators can manage all endpoints through one computer by implementing Faronics Core.

5) Anti-Executable presents a security dashboard to help IT staff monitor exceptions and violations of endpoints. The dashboard includes information

[6] Bit9 Cloud Services https://www.bit9.com/solutions/cloud-services/
[7] Anti-Executable Enterprise http://www.faronics.com/en-uk/products/anti-executable/enterprise/

like top blocked programs, daily and recent violations, top violated machines and violations counter[7]. The dashboard is shown in Fig. 2.2.

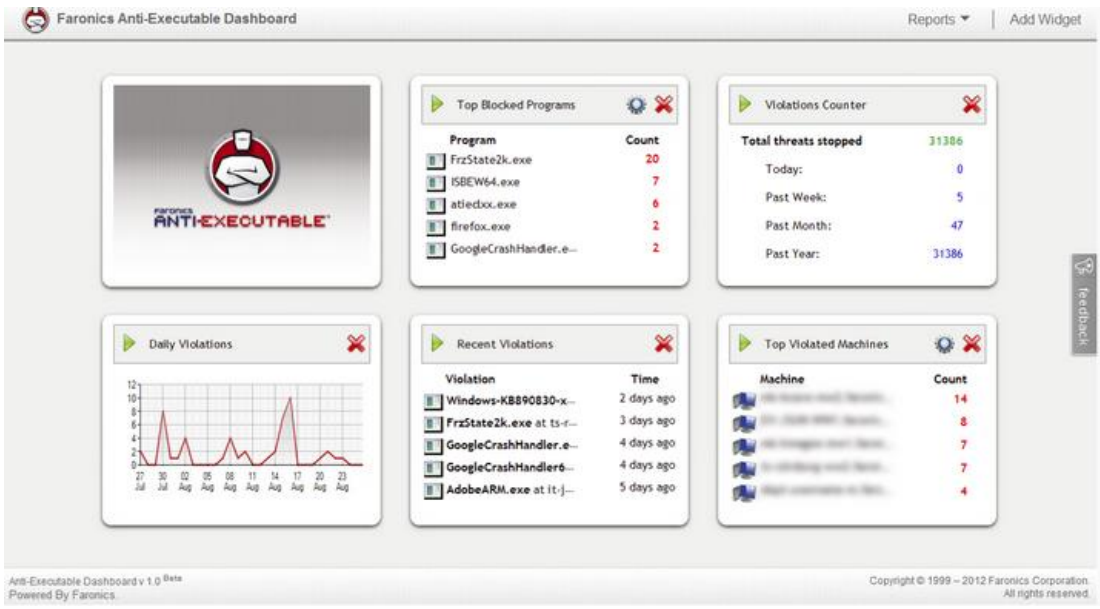

Fig. 2.2 Faronics Anti-Executable Dashboard[7]

The working mechanism is when an application attempts to run, Anti-Executable will check the digital signature. Once it is verified, Anti-Executable will compare it to publishers stored on the white list. If the publisher of the application is involved, it will be permitted to execute. If the application is not signed, Anti-Executable will calculate its hash value, and then scan hash values on the white list. If same value is found, it will be allowed to run. Otherwise, this attempt will be blocked.

2.3 CoreTrace Bouncer

Same as Bit9, CoreTrace is a computer security company and solely focuses on investigating application whitelisting. Its product Bouncer protects enterprise endpoints by preventing unauthorised applications from executing. Bouncer is available to be used in various fields, from companies whose computers are used by professionals and experts, to other companies such as call centres and ATM machines. As one of the application whitelisting pioneers, Bouncer provides multiple functions to not only address the weaknesses of blacklisting but also overcome the shortcomings and limitations of whitelisting technique. The functions and features are described below:

1) CoreTrace Service provides a database including all known good and known bad executable code. It will combine this information with the behaviours of each endpoint and then present it to IT staff. Thus, IT staff can

have a clear knowledge of the trustworthiness of applications running on endpoints, besides, IT staff can also realise which applications are more popular among employees[8].

2) Bouncer takes security measures and encryption technique to protect itself from attacks, such as malicious modification of the white list[8].

3) Bouncer proposed Trusted Change policy to make whitelisting dynamic and changeable without much effort from IT staff. After IT staff defines their "trusted circle", such as trusted publishers, trusted patches, trusted executable files and trusted file paths, all of these can be changed by employees without further approving from IT staff. "Trusted circle" can even include users, which will be interpreted next[9].

4) IT staff can assign employees to BlockQ and AllowQ to define trusted users. For people who are categorised in BlockQ, they do not have the right to run applications which are not on the white list. Instead, they need to fill in a popup to explain the reason for using this application and wait for response from IT staff. For employees who are assigned to AllowQ, even though they have to submit reason likewise, they can temporarily run the application before receiving response of permission or refusal[10].

CoreTrace concentrates on designing and implementing Bouncer in order to make it have attributes of "high-security and easy-change"[9]. Its application whitelisting technique ensures only authorised application can be executed. These whitelisted applications include those who are authorised when Bouncer initially installed and those who obtain permissions later when they satisfy the Trusted Change policy.

2.4 A behaviour-based prototype program

Apart from researching existing application whitelisting and whitelisting products which are published by computer security companies, I also studied many papers which are related to whitelisting approach and behaviour-based technique. In this part, a behaviour-based prototype program named Tracer will be discussed (Kim, et al., 2010). This program makes contribution to the

[8] Solution Overview: CoreTrace Bouncer
http://www.smartinfosec.com/phocadownloadpap/coretrace_brochure_bouncer_overview.pdf
[9] BOUNCER by CoreTrace, Lower TCO with more secure, more available endpoints
http://www.maple5.com/coretrace.pdf
[10] CoreTrace Bouncer Improves Application Whitelisting http://www.darkreading.com/risk-management/coretrace-bouncer-improves-application-whitelisting/d/d-id/1091190

methods of monitoring processes of an unknown executable file and analysing the malignancy of the file based on its processes.

Unknown executable files are executed on a virtual machine. The virtual machine is isolated from the Internet, but connected to a specific network, which is solely for the experimentation. This is because some malware do not implement malicious actions if it is disconnected to network.

When an unknown application runs, process will be created. Tracer will collect these process names and process IDs and monitor abnormal behaviours based on these process IDs. Only behaviours which are caused by these specific processes will be recorded, behaviours which are related to other applications will be ignored. Here, if a process generates another process, the new process ID will be collected and traced likewise.

In summary, for identifying whether an unknown executable file is malware or not, Tracer will run it on a virtual machine. Tracer will collect all IDs of processes related to this application and monitor abnormal behaviours of these collected processes, Tracer records this information. When execution is terminated, Tracer will show the test result, which contains abnormal behaviours and their happening time.

3 Specific background research

In the previous chapter, some whitelisting products published by computer security companies are searched and described, in addition with an academic research of behavioural detection program. Some techniques they applied, such as virtual environment for malicious behaviour detection, hash value for application confirming, are critical and essential to some extent. In this chapter, critical techniques and knowledge which will be implemented in my project whitelisting software are introduced. These include hash function, digital signature and certificate, hooks and sandbox.

3.1 Cryptographic hash function

Cryptographic hash function is an algorithm which can produce a fixed length hash value by entering a message with arbitrary length. The key features of hash function are that it is computationally infeasible (1) to have two messages producing the same hash value, and (2) to calculate a message which maps to a specific hash value (Stallings, 2011). When the content of a message is altered, the hash value will inevitably change. Thus, cryptographic hash function is a prevalent method for checking the integrity of files, data and

codes. Currently, MD5 and SHA-1 are popular and commonly used hash algorithms.

Message authentication code (MAC) is an application of cryptographic hash function. It extends hash functions by adding a technique of symmetric encryption. MAC is usually used for exchanging messages between two entities, both of whom possess the shared key for message encryption and decryption (Stallings, 2011). MAC is sent with its corresponding message and provides the verification of message integrity.

For the project of whitelisting software, I chose hash functions instead of MAC. The reason is: MAC is used to check the integrity of transmitted messages. As hash functions are public, without encryption, attackers can change the message and calculate the new hash value. But if the hash value is encrypted (which is MAC), attackers cannot generate the MAC as they do not have the secret key for encryption. However, for whitelisting, applications and their hash values are isolated from each other. When an application is infected, it does not affect its corresponding hash value stored on the white list. When this infected software attempts to run, a new hash value will be produced and compared to the value on the white list. Such, there is no need to use keyed hash function, i.e. MAC.

3.2 Digital signature and certificate

Digital signature is another authentication mechanism that applies cryptographic hash functions. However, unlike MAC, which uses symmetric encryption for hashed message, digital signature is produced by firstly calculating the hash value of a message and then asymmetrically encrypting the hash value using the creator's private key (Stallings, 2011). A digital signature has two main characteristics: (1) it provides the authentication of message integrity. Same as MAC, without the knowledge of the key, it is computationally impossible to modify a message and produce its corresponding digital signature. (2) It helps people to verify that the message is exactly from the claimed creator. This is because only the creator possesses the private key, thus if the message is properly signed with the correct private key, it is signed by its creator (assuming the private key is safely kept and is not stolen).

Typically, a certificate involves the owner's information and its public key. As mentioned before, digital signature is performed by applying asymmetric encryption algorithms with a private key. For asymmetric encryption, there is a pair of keys which are a private key and public key. A message encrypted by

either of the keys only can be decrypted by another key. Owners keep one key confidentially and release another key publicly. The former is called private key and the latter is public key.

Digital certificate is the corresponding part of digital signature. Certificates are issued and signed by certificate authorities, which are third parties trusted by public. People can verify the validity of certificates by checking the signature signed by certificate authorities. Then, people are able to abstract the public key and use it to authenticate the integrity and source of messages.

In terms of application's digital signature, or code signing, the same mechanism is applied. After obtaining certificates, publishers sign their software by using the private key and create a signature file containing the certificate[11]. In my project, some trusted publishers will be predefined; all applications which are signed by these publishers can be automatically trusted. It is achieved by authenticating their signatures against the trusted publishers' certificates.

3.3 Windows hooks

Windows operating system applies event-driven mechanism. Usually, applications can only handle events which are related to it. For example, an application will give a response when it receives messages from other applications or when it is double clicked by users, but it shows no actions when other unrelated application runs. However, by implementing hooks, applications are able to intercept events generated by other entities and conduct further operations.

In Windows, there are various kinds of hooks; each kind of hook handles a specific aspect of event, for instance, WH_KEYBOARD hook enable applications to monitor keyboard input[13]. An application with a hook has the ability of intercepting a message which is sent to a designated destination or a specific event which is handled by the applied hook. After intercepting this trend, the application can act on the message. What action can be taken is based on hook type. For some types of hooks, applications applying them can alter the message or even terminate this process. While for other types of hooks, applications can only monitor the message without any further actions[13].

[11] Introduction to Code Signing http://msdn.microsoft.com/en-us/library/ie/ms537361(v=vs.85).aspx
[13] Hooks Overview (Windows) http://msdn.microsoft.com/en-us/library/windows/desktop/ms644959(v=vs.85).aspx

For the project of whitelisting software, the implementation of hooks into the software is a fundamental procedure. When an application attempts to run, the whitelisting software needs to firstly stop this behaviour. Then, the application will be inspected against the white list to identify whether it is authorised or not. Here, in terms of stopping the execution of the application, an intercepting function needs to be applied. This intercepting function should be able to monitor the appearance of a new process and temporarily stopping it. Hooks can be an efficient method to achieve this function.

3.4 Sandbox

A sandbox is a mechanism which provides a kind of virtual machine enabling programs to run separately from the real system. Sandbox applies a series of security policies, thus, applications running on it can only affect the sandbox instead of the real system and other simultaneously running applications (Dalcher & Teddy, 2013). To be specific, sandbox provides a tightly restricted environment by implementing a variety of security policies. These policies can either control the behaviours of applications running on it, such as blocking a behaviour which may cause system crash (Wen, et al., 2012); or hook the API and partially duplicate critical system components and files, such as system registry and directory[14]. For the latter, when applications try to alter these critical files, they merely access and modify these copies not the real files. Thus, the real files are still kept safely and the real operating system is not compromised.

Emulation is another kind of virtual technique. Unlike sandbox, which utilises the real system resources, emulator creates an entire virtual environment[13]. This means emulators copy the entire computer environment, such as memory management system, operating system and API calls. Then, when executing an application in this virtual environment, the behaviours of the application can be observed, the malicious actions can be identified while the real system is unaffected.

Regarding the project of whitelisting software, as introduced in the first section, it includes a technique of behaviour-based detection. In order to monitor the characteristics and patterns of an unknown application, and simultaneously prevent possible malicious behaviours from injecting the system, the previously mentioned techniques can be imitated to build a virtual environment.

[14] The evolution of technologies used to detect malicious code
http://www.securelist.com/en/analysis/204791972/The_evolution_of_technologies_used_to_detect_malicious_code#plusminus

4 Project progress

This chapter mainly focuses on the project of whitelisting software. The interim product will be described firstly. The architecture and flow chart of the project are shown in order to clearly illustrate the components and functions of whitelisting. This is followed by a report of the project progress, in which the progress is demonstrated in terms of components and techniques.

4.1 Interim product

The whitelisting software is designed to achieve some fundamental functions. As mentioned in the part of introduction, the main function is that it only allows the execution of authorised applications; applications which are unknown to the software will be rejected or blocked at first. Apart from that, the software is designed with some other functions which enable it more practical. These additional functions are: (1) behaviour-based detection, which aims at inspecting an unknown executable file and then providing the behavioural description of it; (2) blacklisting prevention, which means applications on the black list will be blocked even they are on the white list. In this section, the architecture of the software and the components in the architecture will be interpreted, followed by a prototype description with a flow chart which demonstrates the process of the whitelisting software.

4.1.1 Architecture of whitelisting software components

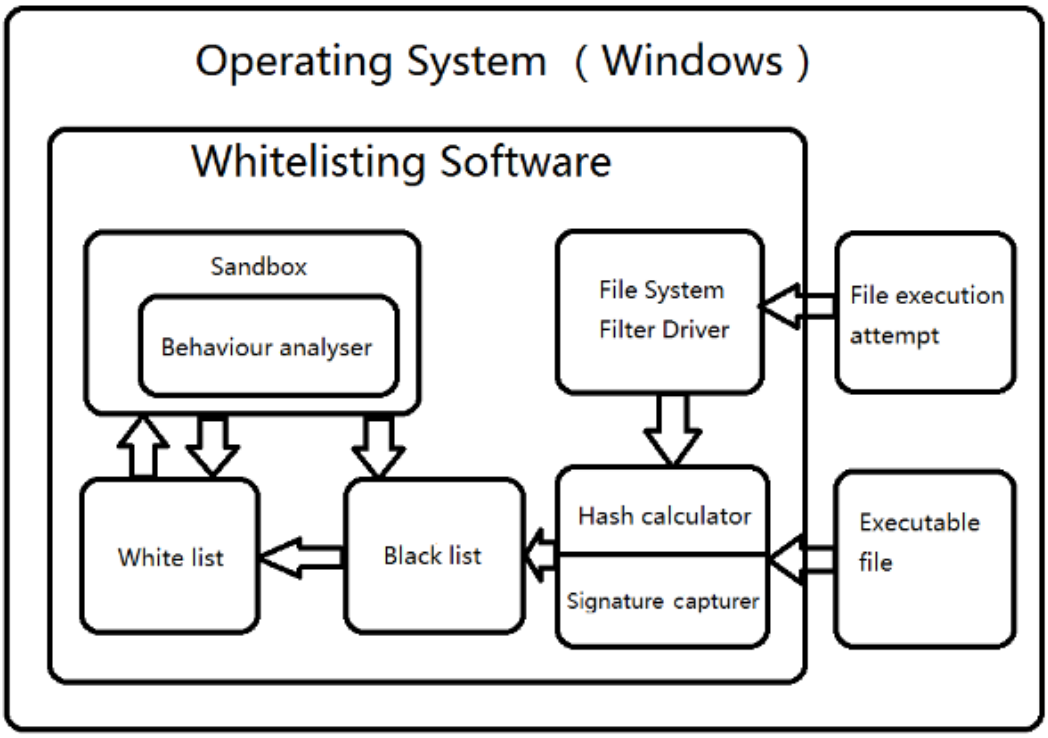

Fig. 4.1.1 Architecture of whitelisting software components

Fig. 4.1.1 depicts the architecture of the whitelisting software. From it, components the software includes and the relationship among these components can be seen.

The whitelisting software is performed under Windows operating system, its component parts are File System Filter Driver, hash calculator and signature capture, virtual sandbox, behaviour analyser, white list and black list. In order to achieve its functions, i.e. detecting the execution attempt of applications and either permitting or blocking the attempt, the whitelisting software needs to communicate with some components which are outside the software, these components includes file execution attempt and executable files.

File execution attempt is a component part which is on the Windows platform. When an application is going to run, the file execution attempt appears. It is a request sent by the application to a targeted file system.

File System Filter Driver is a kernel-mode component part which is able to modify the behaviour of a file system[15]. It makes use of hooks to intercept requests or messages, then, modification, prevention and other manipulations

[15] What is a File System Driver? (Windows Drivers) http://msdn.microsoft.com/en-us/library/windows/hardware/ff557282(v=vs.85).aspx

can be performed[16]. For the whitelisting software, the File System Filter Driver will intercept the File execution attempt, i.e. the request targeting at a file system in order to run the application.

The component of hash calculator and signature capturer can communicate with File System Filter Driver, executable file, black list and white list components. After being informed the interception of file execution attempt from File System Filter Driver, this component part will communicate with the executable file, calculate its hash value or abstract the file's signature.

Black list stores the hash values of executable files which are either previously detected malicious code by behavioural detection or unwanted legitimate software, such as games. Black list has the priority over the white list, if an application is on the black list, it will be blocked even if its hash value or publisher's certificate is on the white list.

The white list component includes pre-stored trusted vendors' certificates and hash values of wanted applications. Here, the hash values are calculated by the whitelisting software when applications are authorised. They are different from the digests encrypted in signatures. The white list component can communicate with the component of sandbox and inform it applications needed to be inspected.

Virtual sandbox provides a virtual environment. The introduction of sandbox can be viewed in Section 3.4.

The component of behaviour analyser is involved in the sandbox component. Its function is monitoring behaviours of applications which are running in the sandbox. It will detect and record these malicious behaviours.

[16] Introduction (Windows Drivers) http://msdn.microsoft.com/en-us/library/windows/hardware/dn641617(v=vs.85).aspx

4.1.2 Whitelisting software prototype description

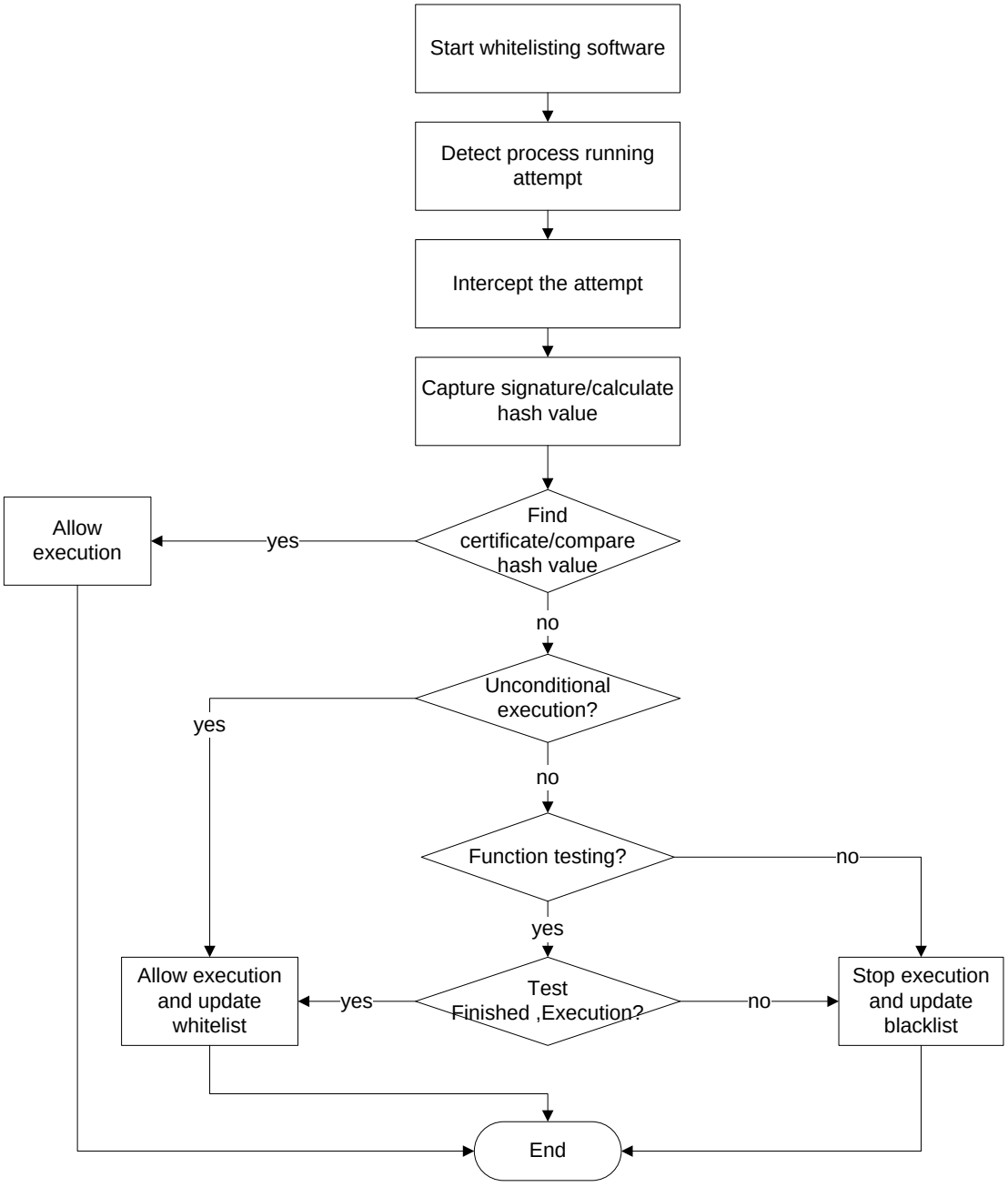

Fig. 4.1.2 Whitelisting software flow chart

As the flow chart Fig.4.1.2 shown, first of all, the whitelisting software needs to be installed and executed. After that, the software can perform its functions and protect its host computer.

When an executable file attempts to run, the whitelisting software will detect this process and intercept this attempt by using the technique of File System Filter Driver.

After that, the hash value of the application will be calculated. Then the black list will be firstly check to see whether this application is on the black list as it has the priority over the white list. If the hash value does exist on the black list, the execution of the file will be refused; otherwise, the white list will be checked.

In terms of the white list, two cases need to be considered in terms of judging the privilege of the file. One case is checking the previously calculated hash value against the white list to identify whether the file is on the list and whether the file is modified. If the same hash value is found on the white list, the executable file will be permitted to run. If not, another case appears. In order to identify the privilege of the file in this case, the signature of the file needs to be captured and the corresponding digital certificate will be searched through the white list. Afterwards, if the corresponding certificate can be found on the white list, it will be used to check the integrity of the application. Then the application will be allowed to execute or stopped depends on the inspecting result.

If the file is rejected to run, the user will be asked to choose one operation from three options: (1) unconditional execution, (2) function testing and (3) stopping the execution of the file. The whitelisting will respond differently depending on the chosen option which will be described in the following paragraph.

When the user decides to run the application, i.e. chooses the first option, authentication will be performed. Only administrator has the right to run applications which are not on the white list. If authentication succeeds, the hash value will be added to the white list automatically. When the user chooses the second option, function testing, behavioural technology will be implemented to monitor the behaviours and patterns of the file. To be specific, behaviours like which API the file called and which registry keys it modified can be monitored and recorded without harming the host device. After testing, a directory containing all the testing information will be demonstrated to the user. Based on it, the user can identify the trustworthiness of the file.

Next, the whitelisting software will ask a user’s decision. If the user determines to run it, same as before, authentication needs to be performed. If not, the response is same as option 3, the application will be rejected to execute by the whitelisting software.

4.2 Progress

The previous section presents the components of whitelisting software and its flow chart. In this section, I will demonstrate the progress of my project. The progress is described separately based on critical components and techniques, such as File System Filter Driver, hash calculation and signature verification.

4.2.1 File System Filter Driver

As mentioned before, File System Filter Driver can modify the behaviour of a file system. Thus, it can be used to intercept the execution attempt of an application. First of all, the driver needs to be created. Then, the function of intercept is able to be achieved. For this part, the driver is under being built, and I will describe the steps of creating this driver as it does not present an interface.

After downloading the Windows Driver Kit and set the environment variable %WINDDK%, I wrote the main function which sets the entry point of the driver, IRP (Initial Receiving Point) and Fast-IO dispatch tables. Besides, call-back registration is done, which aims at tracking changes of a file system. Then, individual function is programmed. In terms of IRP dispatch function, it includes two IRP handlers; one is for passing requests to other drivers and the other is for capture file names. Fast-IO dispatch function is similar to IRP dispatch function; a difference is it isolates this driver from the file system. For the call-back function, it achieves attaching and detaching to a file system based on the system's status (active or not). Thus, the driver can be created.

After the creation of the driver, interception of requests from user-mode to a file system in kernel mode needs to be achieved. Then, the execution attempt of application can be intercepted successfully. This function is undertaken and needs to be further developed.

4.2.2 Hash calculation of executable files

As mentioned before, comparison of hash values on white list or black list with new calculated hash values of applications is a major method to identify whether applications are authorised or not. Therefore, a function to calculate hash values of application and to compare two values is necessary. By using System. Security. Cryptography, it is achieved.

I used two executable files to test the hash algorithm function. One file is Chrome; the other is QQ, which is a popular online chat. As can be seen from the following figures Fig. 4.2.1 and Fig. 4.2.2, for different executable files, the

hash values are different. Apart from that, each file is tested for several times, the calculated hash values are same. Thus, it can be verified that the hash function is programmed successfully.

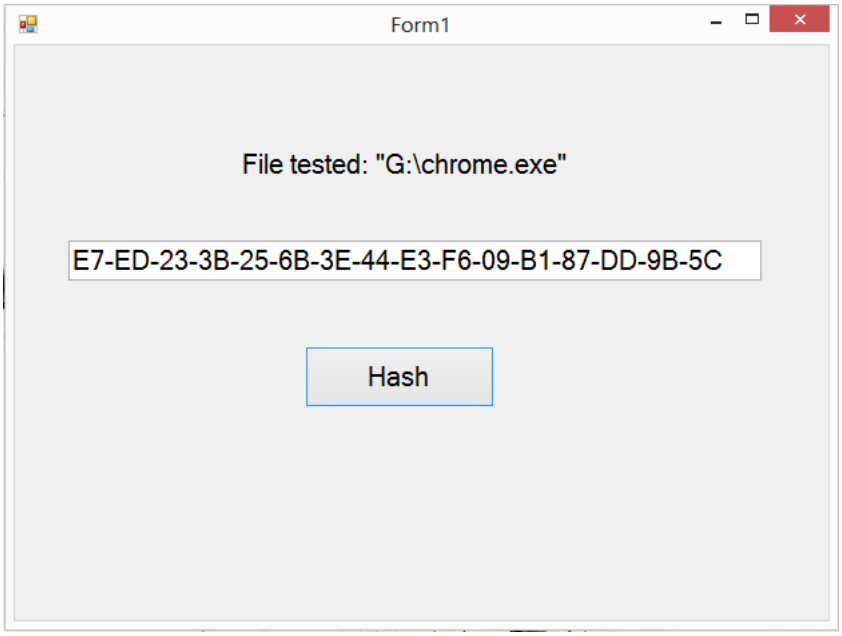

Fig. 4.2.1 Hash value of Chrome.exe

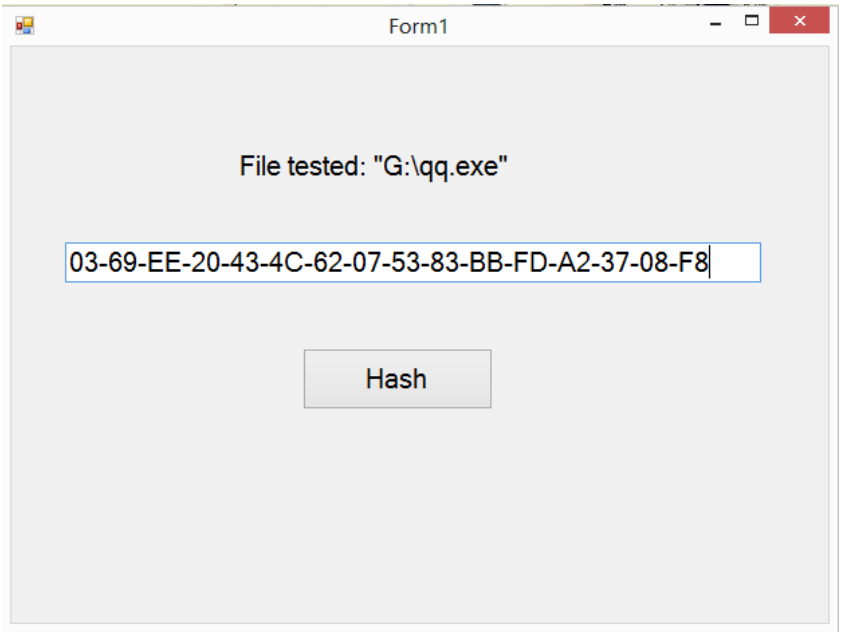

Fig.4.2.2 Hash value of QQ.exe

4.2.3 Access digital certificate

In order to verify the validity of an application, certificate is needed. To be specific, public key needs to be extracted from the signed digital certificate firstly. The whole process of verification is described in the next section 4.2.4. The function of accessing digital certificate is achieved by using System.

Security. Cryptography. X509Certificates. Fig. 4.2.3 and Fig. 4.2.4 show the test results of Chrome and QQ respectively.

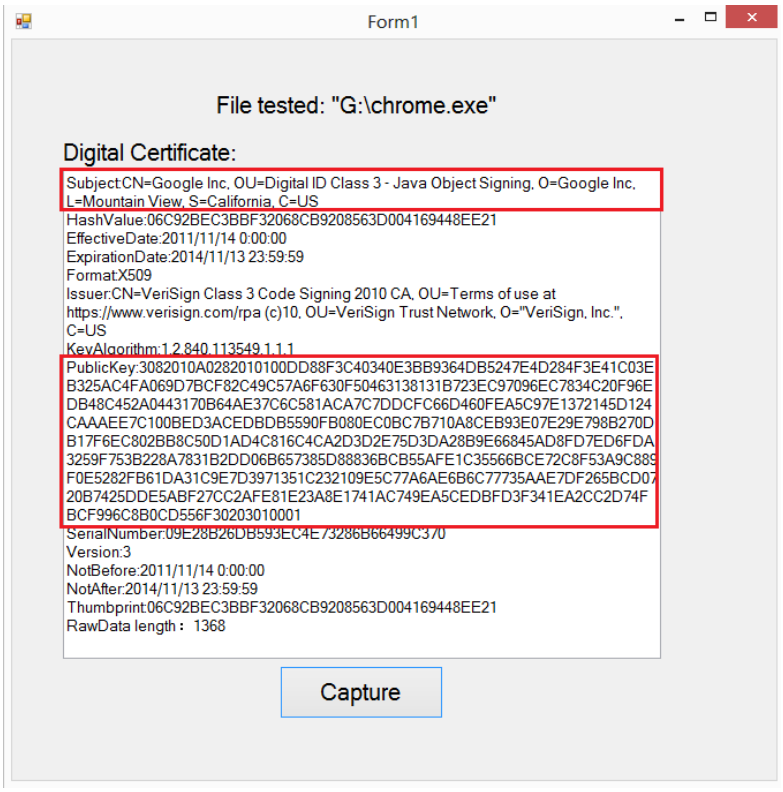

Fig. 4.2.3 Digital certificate of Chrome

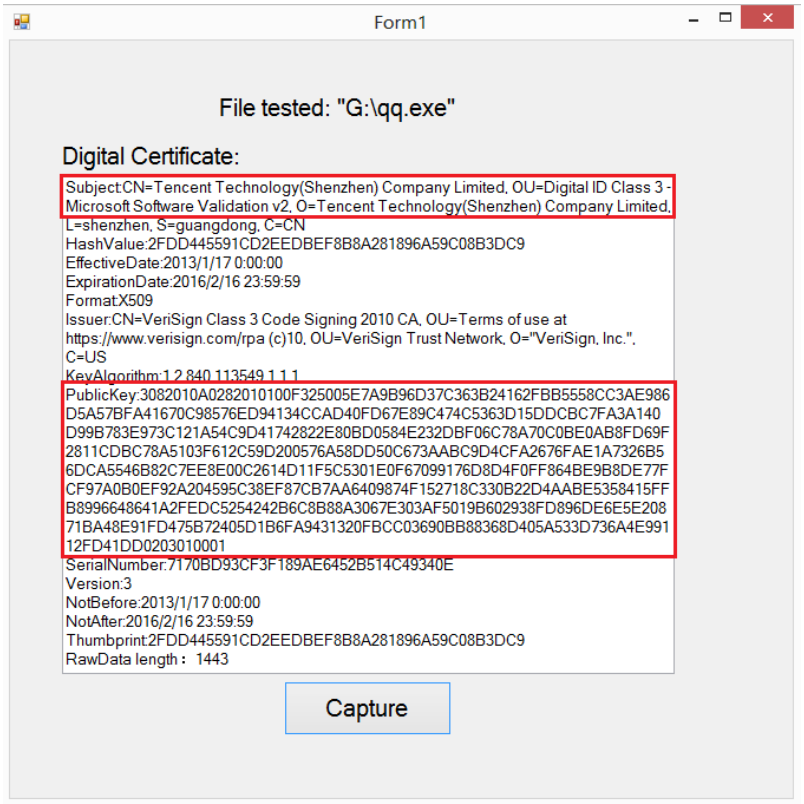

Fig. 4.2.4 Digital certificate of QQ

4.2.4 Verification of digital signature

When an application is executed for the first time, in order to determine the privilege of it, the signature needs to be verified against pre-stored certificates on white list. The mechanism of the verification is described below.

After capture the digital signature of an application, the vendor's certificate will be searched through the white list. If the corresponding certificate is found, then the verification of the origin and integrity of the application will be performed.

Firstly, the whitelisting software will decrypt the certificate by using the public key of the certificate authority. This is because digital certificates are encrypted by certificate authorities. After which, the application publisher's public key can be obtained. This public key will be used to decrypt the digital signature and the digest (the hash value of the application) can be achieved.

Apart from it, the hash value of the binary code of the application will be calculated. This new calculated hash value will be compared with the previous hash value, i.e. the digest. If these two values are equal, this means the application is published by the trusted publisher and is not modified. Thus, it can be identified as authorised application.

For the verification of digital signature, I searched related papers and documents, and understand how it can be achieved in theory. Next, my job is to apply this knowledge into programming.

4.2.5 Virtual environment (sandbox)

In whitelisting software, the virtual environment which is going to be built is not the real sandbox. Since constructing a real sandbox is a big job and it needs a long time to achieve, for my project, a method which utilises File System Filter Driver can work similarly to a sandbox.

A File System Filter Driver is able to monitor messages and requests which are produced by function calls. Before these messages and requests reach their targeted file system, they will be intercepted by the Driver. Thus, these messages cannot touch system kernel and system critical component can be protected.

4.2.6 Behaviour analyser

The function of behaviour analyser is to monitor the characteristics and patterns of an unknown application and then identify malicious or suspicious behaviours. This function can be achieved by observing the appearance of

unexpected or abnormal behaviours of the system. Thus, critical components of the system need to be monitored and logged. These Components include CPU, memory, interface, process, file, registry and network. There are many existing techniques and methods which can be used for tracing files in terms of different components. For example, the behaviours of the first four components can be captured by calling Window API. Techniques which can be implemented for monitoring each component will be described in the following paragraph.

For CPU, pdhGetFormattedCounterValue() can be applied to monitor files' behaviours. GetProcessMemoryInfo() can be used for memory, CALLBACK DebugProc() for interface and CreateToolhelp32Snapshot() for process. In terms of file system, kernel-mode file filter decive driver like Filemon is probably a useful tool for behaviour tracing. With regard to registry, kernel-mode registry filter device driver such as Regmon can be applied. The component of network can be monitored by kernel-mode packet capture device driver like Winpcap.

Same as verification of digital signature, I have the knowledge of how to trace the behaviour of an application and monitor suspicious behaviours. The programming part will be conducted after the achievement of the basic functions, such as verifying signature and implementing sandbox.

5 Project plan

5.1 Research methodology

For the whitelisting software, a main research methodology is case study. In order to achieve the basic functions of whitelisting software, relative research is needed. For example, case study is used for creating a File System Filter Driver. An example of how to create the Driver is searched and studied. Then, a File System Filter Driver can be built based on this research. The implementation of other functions such as signature verification and behaviour analyser can also utilise the research methodology of case study.

Apart from case studies, research methodologies of experimental study and grounded theory can also be applied. Using hash function test as an instance, in theory, the function works correctly if (1) the calculated hash values of different files are different and (2) the hash value of a file is always same when the file is hashed many times. Then, I used Grome.exe and QQ.exe to test the hash function, their hash values are different and kept same for every time of calculation. Thus, it can be verified the hash function is programmed correctly.

5.2 Gantt chart of project plan

The Gantt chart Fig. 5.1 shows the development and plan of the whitelisting software project. Periods which are purple means the work is finished, periods which are orange means this job is under development or are going to be conducted.

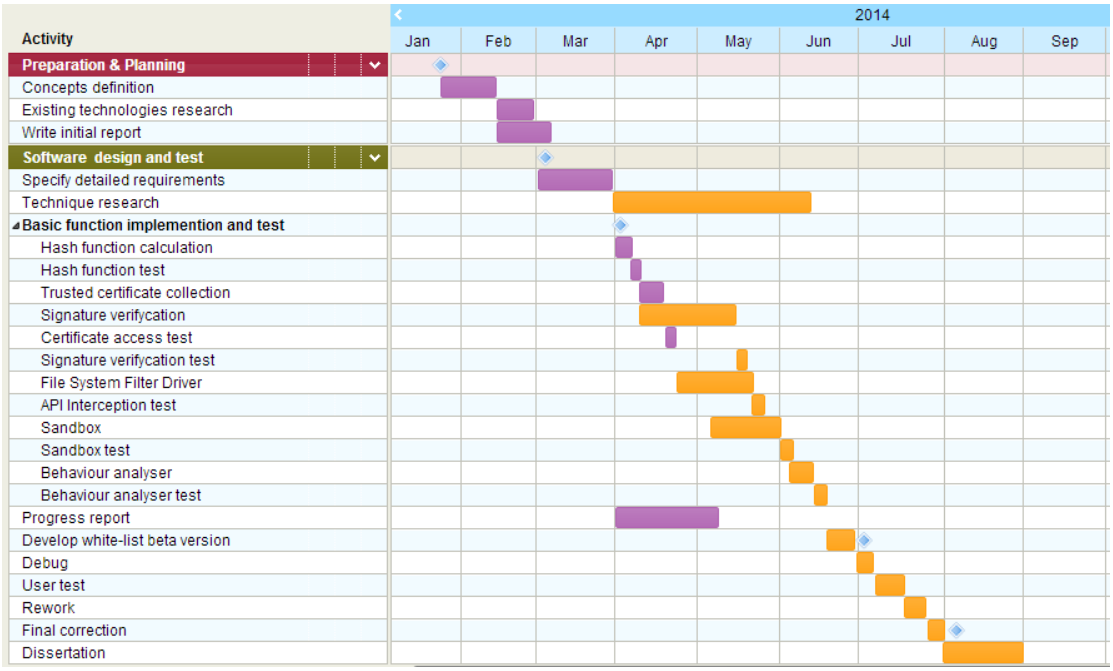

Fig. 5.1 Gantt chart of the project plan

From 24 January to 06 March, I did the preparation and planning of my project. During this period, I searched existing whitelisting products, such as Bit9 and Anti-Executable, which are described in section 2. After that, software design and test are implemented.

After specifying detailed requirements in March, basic functions and tests of whitelisting software are being implemented. These functions include hash function calculation, collection of trusted publishers' certificates, signature verification, accessing certificate function, File System Filter Driver and behaviour analyser. The plan of implementing these functions is described below.

From 01 April to 07 April, hash function was programmed and achieved. This is followed by hash function testing.

From 09 April to 17 April, certificates of trusted publisher are collected, such as Google, Microsoft.

From 09 April to 15 May, the function of authorising an unknown application by verifying its signature is going to be implemented. It was partially done.

The part which has been finished is accessing certificates and obtaining public keys. It was tested on 18 April. The signature verification function will be tested on 15 May.

From 23 April to 21 May, File System Filter Driver is under creation. Then, The Driver will be tested.

From 06 May to 04 June, virtual environment sandbox is going to be created by using Windows hooks, which followed by its function testing.

From 04 June to 17 June, behaviour analyser will be built and function testing will be performed.

From 18 June to 30 June, all of these basic functions will be combined together, new functions such as comparation of hash values will be added. Thus, whitelisting software beta version will be finished.

From 30 June to 04 July, the beta version is going to be debugged.

From 07 July to 17 July, whitelisting software will be tested by volunteers. After using , they will provide feedback.

From 17 July to 24 July, the whitelisting software will be reworked and improved based on the feedback from the volunteers.

From 25 July to 31 July, the whitelisting software will be corrected and improved in order to achieve the final version.

In August, I will focus on writing my dissertation.

Reference

Dalcher, G. W. & Teddy, J. D., 2013. *Systems and methods for behavioral sandboxing.* s.l. Patent No. US 8,479,286 B2.

Gates, C., Li, N., Chen, J. & Proctor, R., 2012. CodeShield: towards personalized application whitelisting. *Proceedings of the 28th Annual Computer Security Applications Conference,* pp. 279-288.

Gates, C., Li, N., Proctor, R. & Chen, J., 2012. CodeShield: Towards Personalized Application Whitelisting. *Priceedings of the 28th Annual Computr Security Applications Conference,* Issue ACM, pp. 279-288.

Huh, J. H., Lyle, J., Namiluko, C. & Martin, A., 2009. Application whitelists in virtual organisations. *Future Generation Computer Systems.*

Huh, J. H., Lyle, J., Namiluko, C. & Martin, A., 2009. Application Whitelists In Virtual Organisations. *Future Generation Computer Systems.*

Kim, D., Kim, I., Oh, J. & Jang, J., 2010. Behavior-Based Tracer to Monitor Malicious Features of Unknown Executable File. *Computing in the Global Information Technology (ICCGI), 2010 Fifth International Multi-Conference on,* pp. 152-156.

Pareek, H., Romana, S. & Eswari, P., 2013. APPLICATION WHITELISTING: APPROACHES AND CHALLENGES. *International Journal of Computer Science, Engineering & Information Technology,* 2(5).

Pareek, H., Romana, S. & Eswari, P. R. L., 2012. APPLICATION WHITELISTING: APPROACHES AND CHALLENGES. *International Journal of Computer Science, Engineering & Information Technology,* 2(5), pp. 13-18.

Stallings, W., 2011. *Cryptography and network security principles and practice.* 5th edition ed. s.l.:Prentice Hall.

Wen, Y. et al., 2012. A Survey of Virtualization Technologies Focusing on Untrusted Code Execution. *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS),* pp. 378-383.