University of Portsmouth

BSc (Honours) Degree in Computer Science

Final Year Project

Debian-based Live Linux Distribution for Penetration Testing

By Josef Abbasikadijani

Project unit: PJE40
Supervisor: Dr. Benjamin Aziz

April 2015

Computing and Mathematics Programme Area

Abstract

Smaller companies cannot afford to hire professional security experts to test the security of the system using penetration testing. There is a lack of user-friendly penetration testing distributions; this project will produce a live Linux distribution based off the popular Debian distribution along with a brand new GUI designed with non-professionals in mind. This distribution will be constructed upon a minimal version of Debian with a stripped down, optimised version of the Linux kernel. A minimalist approach is used to retain maximum functionality with minimal hardware requirements; the user-friendly GUI designed being the basis for user interaction with the system.

The GUI will be developed using the GTK+ library in the Python language using the PyGTK bindings for GTK+. The GUI along with a wide range of BASH scripts will be the basis for the automation of complex penetration testing techniques allowing the layman to simply boot the live distribution off a removable media and perform a plethora of rigorous penetration testing routines, potentially saving companies time and money.

Contents

Glossary

Penetration testing or **pentest** – Intentionally attacking your own system in order to find weaknesses and flaws that could be exploited

***nix** – Unix-like, operating systems (such as Linux) based off the fundamentals and philosophies of UNIX

Debian – Popular Linux distribution with 3 different branches

ArchLinux – Popular modern bleeding-edge Linux distribution

Kali Linux – Currently the best penetration testing Linux distribution, based off Debian

BlackArch – Lightweight penetration testing distribution based off ArchLinux

GUI – Graphical User Interface

Window Manager – An application which handles window functionality and appearance

Desktop Envrionment – A large software package offering applications for day-to-day tasks, includes a window manager, calculator, text editor, login manager etc.

Gnome – A desktop environment based off the GTK+ library

KDE – K Desktop Environment, a desktop environment based off the Qt library

GTK+ - GIMP Toolkit, a GUI library toolkit written for C, it has bindings for almost all popular programming languages. Very popular for desktop distributions and is cross-platform.

Qt – A GUI library toolkit much like GTK+ but written for C++, it uses more memory than GTK+ but has slightly better functionality and is also cross-platform.

OpenBox – An extremely lightweight window manager with lots of configurability and flexibility, currently very popular in minimalist distributions such as CrunchBang

i3 – A tiling window manager, windows are segmented into tiles and fill the entire screen. A far cry from Windows-based GUIs and is hard to learn for the beginner.

BASH – Bourne-Again Shell, the standard shell for all *nix operating systems, also acts as scripting language for said operating systems

DASH – Debian Almquist Shell, a lightweight version of BASH with less functionality aimed towards start-up scripts

Linux Kernel – The core of the Linux operating system, unlike Windows operating systems, modules such as those for sound, graphics, networking etc. are included in the Kernel.

Non-persistence – Technical term for stating that no file will be saved, everything will be reset upon rebooting

Live distribution – A distribution capable of running ‘live’, meaning it requires no installation and will run without issues off a CD or USB

Tint2 – A lightweight Linux taskbar to be used with stand-alone window managers

Swap – A portion of memory reserved to act as memory for when there is not enough RAM, should not be utilized in live distributions

Man – Linux man pages are short for manual pages, giving instructions on all operations possible with a given application

Filesystem – Unlike Windows, *nix operating systems store different parts of the Filesystem in different partitions for performance, security and other reasons. For example, the /home portion could be on a different partition so it can easily be backed up or for added performance.

GRUB – Grand Unified Bootloader, the de-facto bootloader for *nix operating systems

Package manager – Software which installs and configures applications gracefully and in an ordered manner, the package manager used by Debian is aptitude.

Virtual Machine - A type of software which launches and stores an operating system within an isolated virtual environment, great for software development

X Window System – Fundamental windowing system used on almost all *nix operating systems. Comes with a default window manager called X Window Manager but is very rarely used due to being extremely outdated.

1. Introduction

1.1 Overview & Problems

The problem with modern penetration testing Linux distributions is that they are extremely bloated and hard to use for anyone who isn't specialised in IT security. All popular penetration testing distributions are several gigabytes large and are designed to be installed onto the system; many companies cannot dedicate a high performance PC solely for penetration testing and as such require a live distribution capable of running on virtually any low-spec system. Even if the company has the resources to install a penetration testing distribution, using it to the fullest still requires a professional security expert due to the user not being confident in their abilities using the command-line or simply not having enough knowledge of how to perform tests to fully test all aspects of security.

A personal motivation for pursuing this project is experience with smaller companies simply not bothering to perform any penetration testing of their systems which can lead to issues such as loss of sensitive data, unauthorised access to data or denial of service attacks. This is usually not a case of malpractice, but rather a lack of resources for professional services. There is a gap in the market for such a product, by making the distribution as simple and automated as possible, full penetration testing can be performed by the layman.

1.2 Aim

To produce a lightweight, live Debian-based Linux distribution with a user-friendly GUI for the sole purpose of penetration testing. It must be ensured that the distribution is bootable off any removable media, is able to run on virtually any old hardware and that it is very simple to use even for the non-specialised IT technician. By using a GUI developed separately, the user will be able to perform rigorous penetration testing through BASH scripts with the click of a button with the option of running each application individually.

1.3 Objectives

A live bespoke Linux distribution must be constructed based off the stable net installation version of Debian. A GUI will be developed in a given programming language along with BASH scripts. The objectives that must be fulfilled to satisfy the aim of this project are:

- Research and analyse literature regarding similar artefacts/distributions, desktop environments, window managers, programming languages and GUI libraries/frameworks/shells
- Research methodologies and life-cycles, choose and justify the given choice based on project factors
- Interview client, analyse the data gathered and formalise the functional and non-functional requirements
- Design the distribution and GUI separately, making use of the UML. Show how the distribution and GUI tie in together and how they will
- Implement & test the artefact, either move back to the design process if a cyclic methodology is used or evaluate the finished product

1.4 Constraints

The amount of time allotted for this project means that a Linux distribution cannot be assembled from scratch and therefore a fork of the popular Debian distribution will be created which is a popular method even for large organisations. There are not enough computers available to test the compatibility with a large number of systems; therefore it is advised to run the distribution in a

Virtual Machine during the development phase even though it will be converted to a live distribution. All software included must be fully open-source and adhere to the GPL license or similar which allows for non-profit distribution. If a software package is not available with appropriate licensing then alternative options may have to be used even if they are slightly inferior.

This project is extremely time-consuming, especially when dealing with the intricate configuration and fine-tuning of the distribution, as such, much more slack time has been allotted than usual to assure that the project will finish in time. There might not enough time to learn a new programming language for such an ambitious project, there is already some previous Python knowledge and extensive BASH experience but appropriate Python libraries for GUI's still requires learning.

Loss of data is a risk but the damage can be almost fully be mitigated simply by doing regular back-ups, both local backups on an external hard-drive and online backups onto a private FTP server will be performed on a regular basis. The local back-ups will be done twice as often as the remote back-ups as the process is much faster and the remote back-up will simply act as a last resort option. Both the live distribution and the code for the GUI application has to be backed up and backed up separately.

1.5 Report Overview

Section 2 – Literature Review: This is where the bulk of the research is done on similar existing distributions, desktop environments, window managers, GUI libraries and programming/scripting languages used to develop the product. Through critical analysis we can make sense of what direction to head in with regards to design and implementation.

Section 3 – Methodology: Software engineering is a very broad subject and there are countless numbers of methodologies and life cycles that can be followed in order to produce the desired product. Methodologies all have their advantages and disadvantages; it is up to the developer to choose one which is the most suitable which the focus of this section is.

Section 4 – Requirements: In this section the focus of the discussion is how the requirements were gathered from the client in relation to the methodology and also analysing and stating all functional and non-functional requirements.

Section 5 – Design: The design process presented in this section is rather unorthodox but well justified, due to having to design both the distribution and the Python GUI separately. Using the incremental build methodology also makes the design process less linear but the added complexity is worth it in the long run. The design is analysed and critiqued at every step.

Section 6 – Implementation: Once the distribution and the Python GUI has been designed it's time to implement the software. The decisions made on implementation is described in great detail and justified.

Section 7 – Testing: The entire distribution is tested as a whole in this section. The majority of this section will be focused on a table testing each aspect of the distribution and GUI by mentioning what is being tested, how it is being tested, the expected outcome, the actual outcome and comments.

Section 8 – Evaluation: By evaluating the requirements after the product has been designed, implemented and tested, this section gives a final verdict on how well the product stacks up against the initial requirements specification.

Section 9 – Conclusion: This section concludes the entire project and also gives the developer a chance to reflect personally on how the project went. Recommendations are made for future developers seeking to

2. Literature Review

2.1 Overview

There are many existing penetration testing Linux distributions, most popular ones are based off Debian and ArchLinux, however they are all very different. By learning the strengths and weaknesses of different penetration testing distributions we can learn how to create the most lightweight, user-friendly penetration testing distribution. Research will also have to be made into possible desktop environments, window managers, GUI libraries and programming languages available for developing the artefact.

2.2 Existing Penetration Testing Distributions

2.2.1 Kali Linux

Kali Linux is considered the industry de-facto penetration testing distribution (Muniz & Aamir, 2013, pp. 22-23) used by security professionals. The vast amount of books and documentation available simply to be able to use the distribution is a testament to its target market and the sheer difficulty of being able to utilise it. Offensive Security ("Kali Linux Official Documentation", 2015) states that the distribution requires 10GB of hard drive space and at least 512MB of RAM, this is much higher than the distribution presented in this project aims to require and gives further motivation to develop a lightweight solution. The advantage of being able to use such a system is however the largest penetration testing tool set of any distribution, however being able to utilise it requires formal training.

2.2.2 BlackArch Linux

BlackArch Linux is more in tune with the ideologies presented by this project; one major issue for security however is that it is based off a distribution called ArchLinux and the package manager used (PacMan) by ArchLinux does not support package signing through asymmetric key cryptography, this makes the distribution inherently insecure. ArchLinux is also a bleeding-edge distribution, this means that the software has not endured extensive testing and may be unstable and unsuitable for major releases unlike Debian.

2.3 Window Managers & Desktop Environments

A window manager is simply software which controls the functions and appearance of windows by replacing the built-in X window manager; it may come as part of a desktop environment or standalone. Desktop environments are fully fledged desktop GUI software packages which include everything one would need to perform most common tasks. A simple desktop environment can also be assembled manually by combining a multitude of software packages.

2.3.1 Desktop Environments

Gnome and KDE are currently the two major desktop environment available for Linux. Nemeth, Snyder & Hein (2006, p. 758) briefly outline the advantages and disadvantages of both desktop environments; Gnome offers marginally better performance but KDE is more flashy and customizable. Both desktop environments come with very different software; Gnome is based off the GTK+ libraries whilst KDE is based off Qt, these are libraries which allow developers to easily develop GUI applications. It is also stated that it simply comes down to personal preference of the user; however KDE is wholly unsuitable for this project whilst Gnome is a possibility.

It is a fact that Linux has failed to penetrate the desktop market as is shown by Yu & Ramaswamy & Bush (2008, p. 60) where Linux only represents 1% of the total market share of operating systems. Many factors influence the this failure of Linux, however a common criticism is the archaic X windowing system which is the foundation of Linux desktop but lacks in terms of performance and usability as stated by Decrem (2004, p. 52). Unfortunately, no stable, unobscured alternatives exist for Linux and as such it will only be used on desktop computers for niche uses, one of which includes penetration testing.

2.3.2 Window Managers

2.3.2.1 OpenBox

OpenBox is a minimalistic window manager which can either be used as part of an existing desktop environment such as Gnome/KDE or standalone. It is highly configurable and allows for many extensions, window appearances and other window elements. The major benefit of using this window manager is how little RAM memory it uses, as little as 5MB when used standalone. It is extremely common for lightweight distributions to use this window manager and has gained much popularity in recent years.

2.3.2.2 i3

i3 is a tiling window manager, this type of window manager splits up the screen into tiles where the windows reside within them. By doing this, windows cannot overlap and the screen is always full, if a new window needs to spawn a previous window/tile is split into two to make room for a new window. It is a very space-efficient type of window manager and is almost as light-weight as OpenBox, however it is not as configurable and it also is definitely not user friendly to users who have only experienced Windows or Mac OSX operating systems and would be too much of a paradigm shift.

2.4 GUI Libraries

Although GTK+ is the GUI library used by the Gnome desktop environment and Qt is used by KDE, both libraries can be downloaded and used to freely run/develop applications. For example, if a standalone window manager is used, one could simply download the GTK+ libraries and run any applications designed to run in the Gnome desktop environment.

2.4.1 GTK+

GTK+ is an open-source, cross-platform, feature-rich (Krause, 2007, p. 21) library coded in C which allows developing and running graphical applications. Although the library is coded in C, there are bindings for almost every single programming/scripting language such as Python, Perl, Ruby, C, Java etc. A common criticism of GTK+ is that the documentation is inadequate (Krause, 2007, p. 21) and developing applications simply based off the API can be very daunting.

McLean (2005, p. 6) states that GTK+ applications can be very messy and tedious to code from scratch so an additional toolset called ‘Glade’ can be used for producing the layout aspect of the application. Using Glade allows the developer to easily produce an application very similar to the design and generate code which is readable and easy to work with.

2.4.2 Qt

Qt is the main competitor to GTK+, it is also open-source and cross-platform. The Qt library is however coded in C++, this will only be relevant if the application is coded in that language as it also has bindings for a wide range of programming/scripting languages much like GTK+. The differences between GTK+ and Qt are mostly semantic and aesthetic, they have been very different in the past but have homogenised greatly over the years.

Qt is plagued by the same issues as GTK+ when it comes to proper documentation of the API and this is a major hurdle for anyone trying to create GUIs in Linux. This issue might even be a big reason for why Linux has not had success on the desktop platform as GTK+ and Qt are the only two popular libraries for creating GUIs. Dalheimer (2002, p. 11) mentioned that “Until the first edition of this book, there was not much Qt documentation for beginners.” Documentation available for the Win32 API is vastly superior to any documentation available for Linux based GUI libraries/toolkits.

2.5 Programming/Scripting Languages and Shells

2.5.1 Perl

Perl is a high-level, interpreted scripting language; it is a very mature language and has roots in Unix. The coding syntax is very flexible much like C/C++ and due it being a very mature language, there are countless numbers of modules/libraries available for use which makes it an extremely powerful language, especially for larger projects. An issue with this language is the difficulty of learning it for beginners; it is a very old language and as such follows many out-dated programming conventions

2.5.2 Python

Python, much like Perl, is a high-level interpreted language and although they seem very similar on the surface they are very different. Python's syntax is not as flexible as Perl's, indentations are forced and the syntax is set in stone. This can be considered both good and bad; it makes the code very easy to read and also reinforces good coding habits which are especially good for beginners.

Python has gained much popularity in the education field; schools teach students the fundamentals of programming with Python as it forces the programmer to write with proper syntax, rather than being an added bonus as with other languages.

2.5.4 BASH

BASH (Bourne-Again Shell) is the default shell in most *nix operating systems, it is a UNIX shell and primary method performing commands in any *nix system. It can also read/write files and read commands from a file; it also acts as a scripting language.

Because of how the shell is tied into the Linux operating system, the scripting language is extremely powerful and allows for some very impressive elegant code. *nix applications are usually very simple and serve only one purpose, it is up to the scripter to be able to combine the outputs of many different applications and operators to create truly powerful scripts. The Windows equivalent of BASH is PowerShell and the first stable release was in 2013, *nix operating systems have a huge advantage when it comes to shell scripting and anything command-line based.

2.5.5 DASH

DASH (Debian Almquist Shell) is a modern, lightweight shell used in a lot of Debian-based distributions. The main advantage of this shell is faster script execution; this allows the operating system to boot up and run services faster. It does not however have the same amount of functionality as BASH due to it being a more modern shell, so BASH is still the standard Linux shell and DASH is almost always used simply for running start-up scripts.

3. Methodology

3.1 Life Cycles

With any modern software development projects, it is important to adhere to strict to a form of software development life cycle (SDLC) and approach the project is a structured and methodical manner. This section will outline the advantages and disadvantages of each model in regards to this project and find a suitable methodology to follow.

3.1.1 Evolutionary Prototyping Model

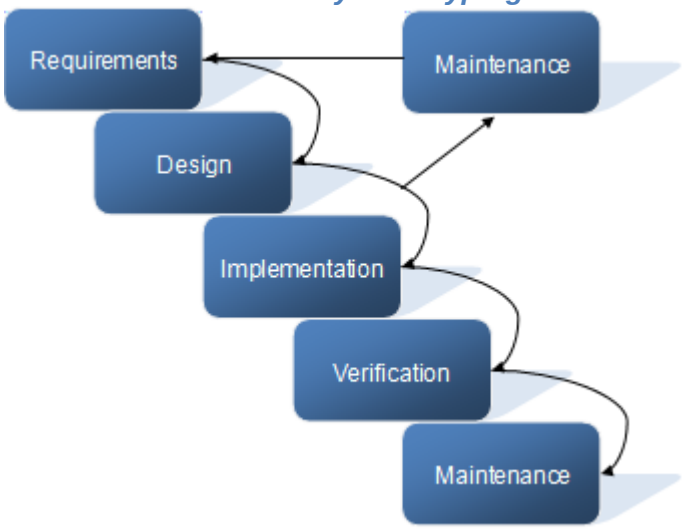


Figure 1: Evolutionary Prototyping Model

The basics of the prototype model is very simple, a prototype is produced based off the initial requirements and design. This prototype can be shown to the client in order to get some feedback regarding changes that could be made to improve it, it can also be analysed by the developers to reveal requirements that were overlooked initially. Prototype modelling is a very modern approach to software development; it also allows the client to be more involved in the entire process as miscommunication or failure to provide appropriate requirements by the client does not end in disaster.

This approach however, does not come without its inherent drawbacks. Creating a lot of prototypes can be very expensive and time consuming, especially if they are simply thrown away at the end. Developers also tend to get attached to earlier prototypes because of the time spent making it as described by Beaudouin-Lafon & Mackay (n.d., p. 1008), even if it is not appropriate.

There are many variations to this type of model; the one most appropriate one is Evolutionary Prototyping Model. With this model, a very robust prototype is produced which is then the basis for the final product, it is constantly refined as needed which is advantageous in an ever changing security field.

3.1.2 Waterfall Model

The waterfall model is one of the simplest type of SDLC, the reason it is called a waterfall model is because the flow of tasks flows downwards in a one-way fashion. This model is good to use when the requirements are well understood and do not change often as once each step is completed, they are not revisited unless it is highly critical. The rigidity and inflexibility of the waterfall model can be a huge problem for larger projects where the requirements are more dynamic.

Smaller projects such could benefit from using the waterfall model as it is suitable

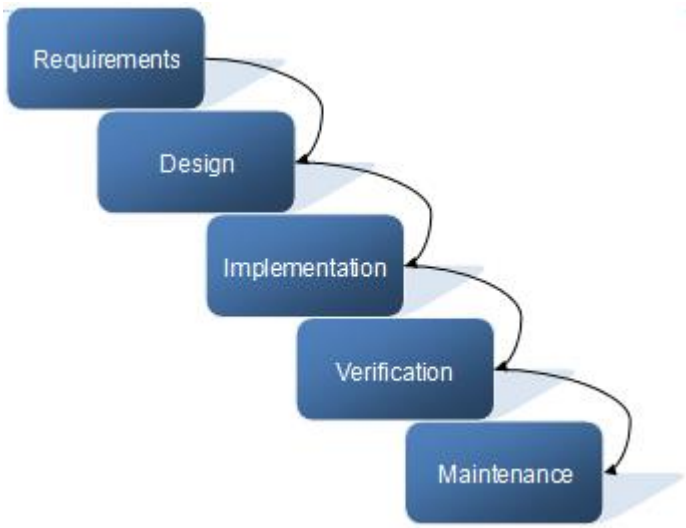


Figure 2: Waterfall Model

for smaller projects and the high risk nature of the model is irrelevant if it is not a critical project. One major drawback of using this model is that it is not really designed for constant updates which this project requires, it is important that the software is updated at least monthly to keep on top of emerging threats and attacks. Another drawback is that the requirements are set in stone and cannot be altered after the requirements stage, but on the flip side, if the requirements are updated later in the requirements stage, it is much more cost effective to rectify the error early on as opposed to other models where a prototype which is now useless has already been created.

3.1.3 Incremental Build Model

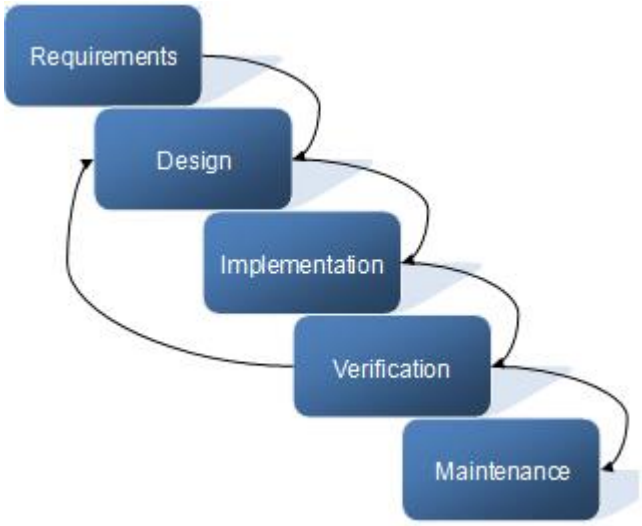


Figure 3: Incremental Build Model

Larman & Basili (2003, pp. 2-3) considers incremental/iterative models a “modern replacement of the waterfall model” and state that commercial and government organisations are slow to change their standards. The incremental build model incorporates methodologies from waterfall and prototyping models, the requirements are gathered then the software is designed, implemented and tested. Based on the test results, the software is then redesigned and it continues on like this in a loop fashion as shown in figure 3.

The changes made between each design and testing phase is very small, but it is incrementally tested and improved upon. The two main benefits of this model is that a working version of the software is available very early on and it is very easy to test (after the first iteration) because only small changes are made.

Just like the evolutionary prototyping model, it can be very expensive to use this model, especially if a fault in the fundamental architecture of the product is discovered very late.

3.2 Choice and Justification

The agile methodology was not even considered for this project, many aspects only apply to a team of developers working together and more traditional approaches seemed more appropriate. Using an overly ambitious methodology for a project with a single developer is only detrimental.

The final choice of methodology to follow for this project is the incremental build model, the main reason for making this choice is that the distribution and the GUI needs to be developed in parallel and it is important to have a working, bootable version of the distribution so the GUI can be developed alongside it. Many elements need to be tested and re-designed based on how the GUI behaves in conjunction with the penetration testing software and the rest of the distribution. The only real disadvantage is cost, but this is a non-issue for this project.

4. Requirements Analysis & Specification

4.1 Gathering

Requirements are gathered during one-to-one electronic interviews with the client, this is done regularly due to adhering to the incremental build methodology which requires regular re-designing and additional involvement with the client. The product will be tested and re-designed as needed until the client is happy, once a final version of the requirements are gathered, a final design and implementation will take place and from thereon out it will simply be maintained by updating the software as needed to keep up with the demands of security threats.

4.2 Analysis

The below requirements are a combination of functional and non-functional requirements gathered and analysed after much discussion with the client. Each requirement has an associated priority level indicated at the end of it, this is a priority level (out of 10) showing the importance of the requirement and how much focus will be directed towards it.

4.2.1 Functional Requirements

- Boot the Linux distribution without installation (10): The distribution must be bootable live off any removable device or disc such as USB memory sticks, CD/DVDs, etc. This is because there will not always be a spare computer that can be used solely as a penetration testing machine, by making it a live distribution the user doesn't need to devote a dedicated computer for it. This requirement has a maximum priority level as without the ability to boot it live it would simply be a sub-par generic penetration testing distribution.

- Access to a wide variety of penetration testing and security software without any configuration (10): The distribution includes software for: geolocation, network analysis, web attacks, wireless, anti-malware and many other applications. A priority level of 10 is needed here as there should be a good selection of penetration testing applications which come pre-configured or the live distribution will not work.

- Launch any application available in the distribution using the developed user-friendly GUI (10): A GUI is designed and created for the sole purpose of making it easy for the user to perform penetration tests. Many of the applications have very ambiguous and complicated names so it gives the user an idea of what each application does. Each application also comes with supporting documentation to guide the user on how to use it effectively. The reason a priority level of 10 is issued is because any application not available through the GUI will be completely useless as the non-professional won't know of its existence and will simply be wasted space on the distribution.

- Perform automated penetration testing using BASH scripts (8): As the project aims to offer a penetration testing solution to non-professional users, the user might not know how to perform penetration testing routines. The scripts use some simple input from the user and uses those inputs as variables for the penetration testing. The user does not need any deep understanding of security, only some basic information about the system. As the applications can be run through the GUI, this requirement will only be an 8 as the three previous requirements are needed to be able to do anything whereas this requirement simply makes it easier to use.

- Read documentation on how to use the software packages (3): Users may still have trouble understanding how to use the individual applications if the scripts don't cover the functionality required. On the other hand, the GUI should be intuitive enough that reading through documentation should not be a necessity, but a nice feature if time allows for it. A priority level of 3 is to signify that it will be an extra feature added if artefact development goes as planned.

4.2.2 Non-Functional Requirements

4.2.2.1 Performance (10)

The entire distribution (including the developed GUI) should be as lightweight as possible without compromising any of the functionality of the software. The entire distribution should fit on a single CD (<700MB) and it should not require more than 512MB of RAM to run. Boot time should also not be very high; steps should be taken to optimise the boot speed. Without this requirement, the artefact would simply be worse than current popular penetration testing distributions which is why the priority level of 10 is warranted.

4.2.2.2 Legal Issues (10)

Simply by opting to make the distribution based off Linux, it is already ensured that most of the software is licensed under the GPL (GNU General Public License) which allows for non-profit distribution. Legal issues can however crop up in regards to individual security software packages, non-GPL packages will either need to be fully omitted or alternatives need to be used. Failure to provide software with appropriate licensing would result in not being able to release the distribution as it would be illegal; therefore a priority level of 10 is needed.

4.2.2.3 Usability (7)

Usability is a major motivation for pursuing this project as stated in the aims in section 1.2, the software should be so easy to use that virtually anyone able to use a computer can insert a CD/USB and boot up the distribution and make use of the GUI to easily perform penetration testing. The GUI should be as simple and intuitive as possible, the only inputs should be the ones absolutely necessary and the scripts should cover the rest. A priority level of 7 is issued, however this is only because it is relative to the other 2 non-functional requirements which are a necessity to be able to produce the distribution whereas usability is simply a case of more being better, user-friendliness is improved as much as time allows it.

5. Design

This product will be designed using the Unified Modeling Language (UML) which is the de-facto system used for designing software. A use case diagram will demonstrate how the client interacts with the system; however the distribution and the GUI will be designed separately as there is a lot of “behind the scenes” designing involved when creating a fork of a distribution.

5.1 Distribution Design

Only the major attributes will be mentioned in this section to omit a large number of trivial design choices with little effect on the actual implementation.

5.1.1 List of Major Distribution Design Decisions

- Based off 32-bit (i386) Debian ‘Wheezy’ Stable net installation
- Single user Root-only mode
- Non-persistence, forensic mode
- OpenBox standalone window manager
- Tint2 taskbar application
- Stripped away redundant kernel modules (e.g. audio, printers, etc.)

5.1.2 Justification for Distribution Design

The distribution will be based off Debian ‘Wheezy’ (stable) net installation. Debian has three different versions, stable, testing and unstable. The stable version has software which has been tested extensively for years which also has the downside of having slightly older software. Surprisingly, the testing version is very stable but the software has yet to been tested for long enough to make it into the stable version. The unstable version is bleeding-edge; as soon as the developers have released a new version of software it is available for download immediately.

The reason the stable version will be used rather than the testing version isn't solely due to stability but also because compatibility with older hardware is an issue with the testing version. There is a choice between a 64-bit and 32-bit version, almost all of the benefits of the 64-bit version is irrelevant so the 32-bit version is chosen. The only benefits to using the 64-bit version is that more than 4GB of RAM can be used and slightly better performance in some cases, the downside is major compatibility issues with older hardware and incompatibility with a lot of smaller penetration testing applications.

As the only purpose of the distribution will be penetration testing, it must be configured to run in forensic mode. Forensic mode is an informal phrase used by penetration testing distributions for saying it runs in root-only mode and is not persistent in any way, so if the distribution runs off a USB, none of the files on the USB will change and the hard drives won't be mounted so once the system shuts down or reboots everything will be reset.

Running the system in root-only mode is not dangerous for a live distribution as the files cannot be altered anyways and the hard drive is not affected, the benefit of this is total power and the ability to perform any action with any application. A swap partition is a commonly used feature where a portion of the hard-drive is used to serve as additional RAM if needed; this will be disabled because to ensure we are in forensic mode the hard-drive must never be written to or read from. This also increases the need for lightweight options the desktop environment and optimisation of the kernel modules.

After extensive research and analysis as shown in chapter 2, the desktop environment will be an extremely lightweight combination of OpenBox standalone window manager and tint2. OpenBox is a window manager previously mentioned in section 2, it is a highly configurable and full-featured lightweight window manager, and tint2 is a lightweight task bar application. The only real reason to use a window manager for this distribution is to be able to have an environment where the GUI application developed in this project can be run, so it makes sense to use the most lightweight option to increase compatibility with older systems.

There are many redundant modules in the stock Linux kernel which need to be stripped away, for example printing modules and audio modules can all be removed as they are completely useless for this distribution. By doing this, we can increase boot-up speed; decrease RAM usage and decrease CPU load (albeit minor).

5.2 GUI Design

After much research of existing literature and past experiences with programming languages, Python 2.7 will be the language used to develop the GUI. Although there are much newer versions of Python (3 and beyond), Python 2.7 is still in development and used widely due to how different newer versions of Python are and because according to Neckel, (2015), "…not all packages have been ported to Python 3.X". Another major reason for choosing Python 2.7 over 3.x is because of using Python 2.6 in year one of University which is syntactically consistent with Python 2.7.

In order to develop it a GUI library is also needed, the two major choices are Qt and GTK+2/GTK+3. GTK+3 was chosen as the GUI library toolkit, it has excellent Python bindings with vast amounts of resources and documentation available. The GUI design will be very compact and require very little mouse movement as Kostaras & Xenos (2010, p. 435-436) found that badly designed GUIs which required a lot of mouse movement was a major culprit for UEMSD (Upper Extremity Musculoskeletal Disorders).

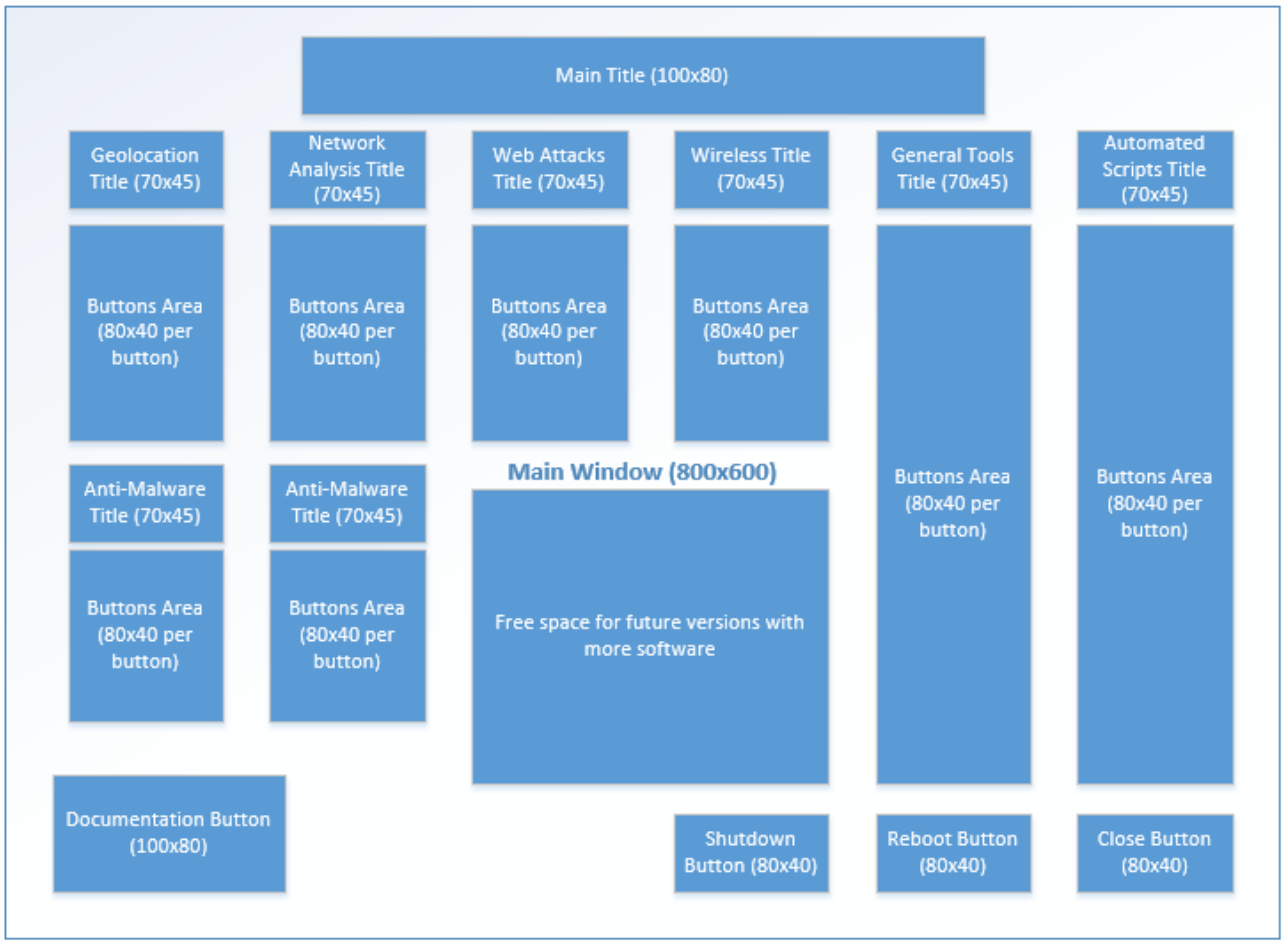


Figure 4: Main interface design

Figure 4 shows how the elements of the GUI will be laid out, the reason the main window is 800x600 is because this is also the lowest resolution that will be supported by the distribution due to older hardware, and as such it has to fit on the screen. The user will have the option of running any application individually (including non-penetration testing applications) with supporting documentation or simply performing automated tests by running the scripts.

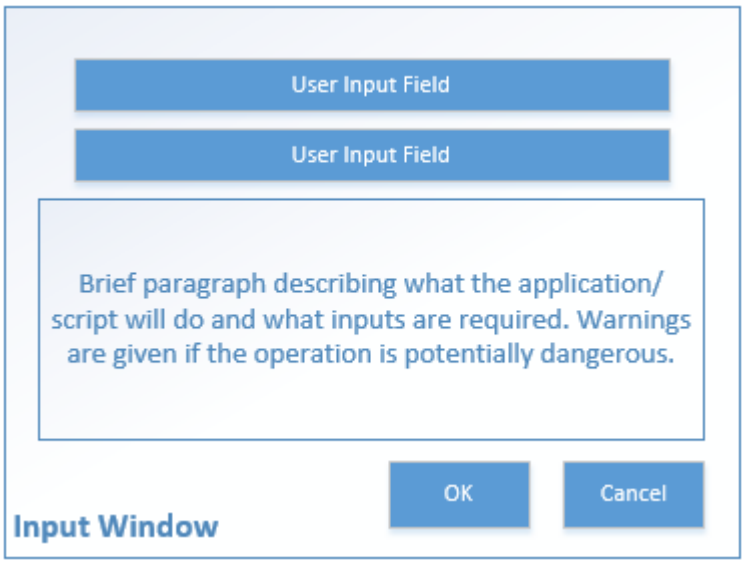


Figure 5: User input pop-up window

Figure 5 shows how the user will input the data required for the applications and scripts. If input is required, the pop-up window as shown will appear and guide the user to input the required data for performing the action. The pop-up window and element dimensions are completely variable depending on the intended purpose.

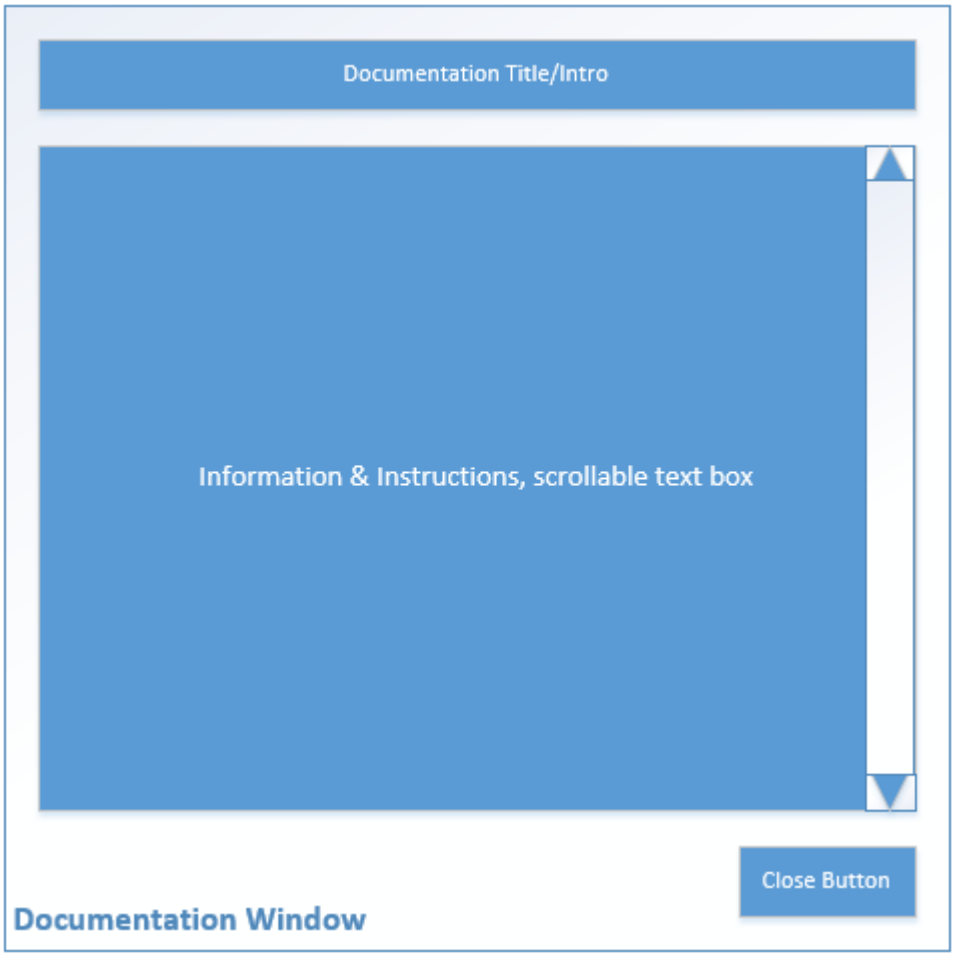


Figure 6: Documentation Window

The documentation window shown in Figure 6 has been revised since the initial design, it was originally planned to be an in-depth document explaining what operations each application can perform but this was unnecessary as Linux includes incredible manual pages or 'man' pages. Instead, the documentation window simply describes each group of applications in terms of their operations and the respective terminal commands for access the 'man' pages. Some information will also be given about the scripts; however it should not overwhelm the user or waste their time.

5.3 Putting It All Together

Figure 7 shows the chain of sequence made by the distribution from boot up to having a fully functional operating system. Of course, trivial and non-functional steps are omitted for clarity as there are many small processes in between each major step. Every step prior to the final step is simply a method for utilising the Linux kernel in a GUI environment.

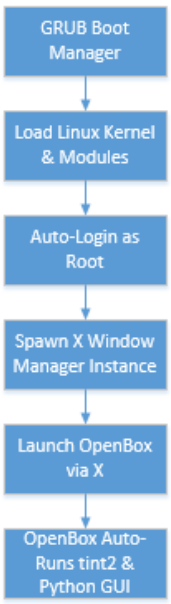


Figure 7: Flow of Processes in Distribution

5.4 UML Use-Case Diagram

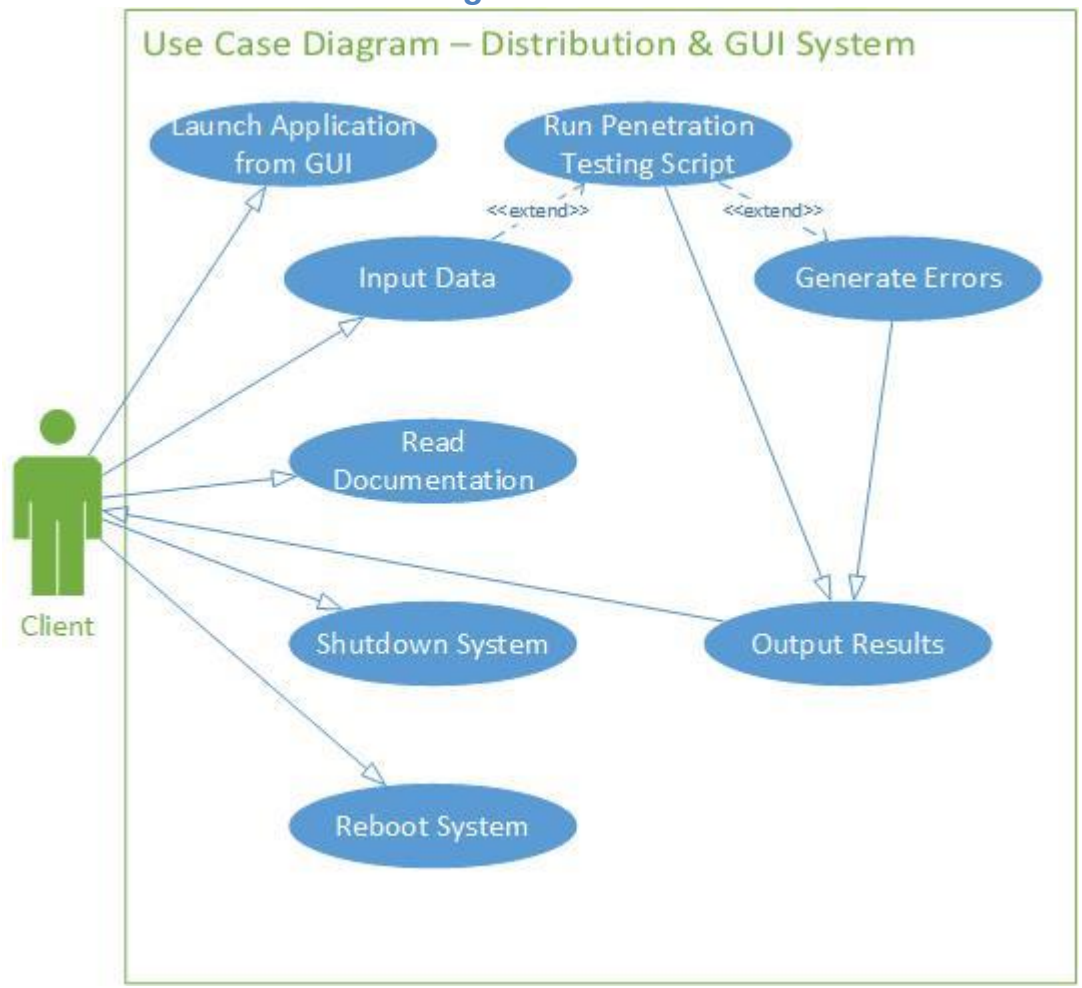


Figure 8: Use Case Diagram

Figure 8 demonstrates use cases for the client; this is how the client (or other users) will be able to perform actions within the system as per UML standard convention.

6. Implementation

First, a live distribution has to be forked from Debian in order to facilitate the penetration testing software and the Python GUI, as well as being able to fulfil the requirements of the client as no other current distribution satisfies the requirements gathered. Debian is a very popular distribution to fork, especially for penetration testing as it has some of the highest quality repositories of any distribution due to how the packages are digitally signed as described by Cappos et al. (2008, p. 12-14).

There was a plethora of changes made to the distribution and listing every element that has changed is not a fruitful endeavour. Instead, the content of this distribution is split up into 3 different main categories, the base, the essentials and the desktop environment.

6.1 Distribution

A network install version of Debian 'Wheezy' stable was downloaded and run through VirtualBox, the reason for doing so is that it allows for more experimentation and risky behaviour which is needed when following the incremental build model. The network install version is an extremely small part of the Debian distribution which only includes the essential parts needed to boot into the system and the rest is downloaded via the package manager aptitude.

6.1.1 The Base

No user other than the default root was created; this is generally not recommended as it may compromise the integrity of the Filesystem. As the distribution is going to run live, none of the changes made to the system will be persistent so running the distribution in root-only forensic mode will be no issue and will allow the user to perform any operation. Auto-mounting was disabled and it is ensured that the hard-drive is completely ignored to ensure that the live distribution has no effect on operating systems such as Windows that might be currently installed.

Partitioning during the development process is simply a matter of convenience, once the distribution has been converted to a live distribution, the initial partition table will be irrelevant. As such, during the development process the distribution was allocated an arbitrary amount of hard-drive space and an atomic partition structure was used, this means that only / (root) was mounted on a single partition. There are a host of different reasons for why this would be a terrible idea for a regular distribution including, but not limited to: security, data integrity, performance etc. but for a live distribution of this nature this approach is optimal.

After the initial configuration, the base system was installed which only includes the Linux kernel, some Debian scripts and basic software needed to run the system. GRUB was chosen as the boot manager of choice; it is the de facto standard used in most distributions due to being both flexible and simple, it will also be the boot manager used for when it is converted to a live distribution.

It was important to convert it to a live distribution before the artefact was fully developed to ensure that it did not break during any part of the process. Each time the artefact was verified & validated using the incremental build model, the entire system was converted to a live distribution. In order to achieve this, the entire system is copied using rsync, isolinux binary/libraries are copied over to allow live booting and then the entire Filesystem is squashed in order to compress it. After this, the ISO file which will be copied over to the CD/USB is generated using genisoimage with appropriate parameters. There were a plethora of intermediate steps in order to achieve optimal results; however the main steps required are shown in figure 9, 10 and 11.

```
rsync -av / myfs/ --delete-excluded --exclude="$work_dir" \
 --exclude="$snapshot_dir" --exclude-from="$snapshot_excludes
}
```

Figure 9: rsync copying filesystem

```
mksquashfs myfs/ iso/live/filesystem.squashfs ${mksq_opt}
```

Figure 10: mksquashfs Compression

```
genisoimage -r -J -l -D -o "$snapshot_dir"/"$filename" -cache-inodes \
-b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 \
-boot-info-table -allow-limited-size iso/
```

Figure 11: genisoimage, .squashfs converted to bootable iso

6.1.2 The Essentials

When choosing what applications to include using the package manager aptitude, it was important to compare the functionality of the applications in relation to the file size and RAM usage of it. Some crucial parts of the system such as the Linux kernel also had to be stripped of many of its features, all packages and kernel modules related to sound; printing and other useless aspects have been completely removed. Extensive documentation of over 800MB was also completely removed; this is documentation outside of the ‘man’ pages and serves no purpose for this artefact as it contains mostly irrelevant details.

6.1.3 The Desktop Environment

Figure 7 shows the flow of processes from booting up the system to the desktop environment being displayed and ready for the user to make use of. This sub-chapter will show an extremely light-weight and simple approach to hosting the developed interface using very little memory.

```
root@debian-up649230:~# cat /etc/inittab | grep root
1:2345:respawn:/bin/login -f root tty1 </dev/tty1 >/dev/tty1 2>&1
```

Figure 12: /etc/inittab contents

The /etc/inittab file was edited to automatically login as root as the first step as shown in figure 12, every time the system boots the root user will now be automatically logged into a tty1 session.

```
root@debian-up649230:~# cat .bash_profile
startx
```

Figure 13: .bash_profile for root

The .bash_profile file as shown in figure 13 describes what actions will be performed once the user has logged through the BASH shell, as root is logged in automatically, this step will also be performed automatically. Startx is a command which initialises the X windowing system; X has already been pre-configured to choose OpenBox as the default window manager as shown in figure 14 which means OpenBox will be launched automatically.

```
root@debian-up649230:/etc/alternatives# update-alternatives --display x-window-manager
x-window-manager - auto mode
  link currently points to /usr/bin/openbox
/usr/bin/openbox - priority 90
  slave x-window-manager.1.gz: /usr/share/man/man1/openbox.1.gz
Current 'best' version is '/usr/bin/openbox'.
```

Figure 14: Command to display the window manager of choice for X

OpenBox will now be launched as root automatically upon boot, the applications developed for this artefact are now auto-started upon OpenBox initialising as shown in figure 15.

```
root@debian-up649230:~/.config/openbox# cat autostart.sh
tint2 &
python ~/dev/pengui/interface.py &
```

Figure 15: OpenBox autorun.sh file

Most Linux distributions do not use this approach and opt for using login managers and full desktop environments requiring hundreds of megabytes of RAM with very little added functionality. The GUI was implemented in this way to both save RAM and also to greatly speed up the boot-up process by ignoring may intermediate steps. This whole process is rather hack-like and crude, it is a common criticism of the X windowing system that the process is very convoluted but this method is the method for getting the fastest boot-up time.

6.2 Python GUI & BASH Scripts

This part of chapter 6 will describe how the Python GUI was developed from the design in chapter 5, how the signal handler receives signals and runs the appropriate functions in the Python code. These functions then use the variables input by the user to launch the BASH scripts through sub-processes and using the inputs as variables.

6.2.1 Development Tools

During the development process, the distribution will be run via a VM (Virtual Machine) as it saves time, resources and with fewer errors. VirtualBox is the software used to run the VM; this makes it extremely convenient to code the Python GUI and BASH scripts. After each iteration of the artefact as per the incremental build model, the distribution (along with the developed GUI) is compiled into a live distribution to prepare it for the testing process and the next iteration of the model.

Both the Python GUI and BASH code is written in a text editor and multi-purpose IDE called Geany, About Geany (2010) states that it is based on GTK+ which not only falls in line with the design choices made for the GUI which is about to be developed, it will also be suitable as a text editor for the distribution itself. The IDE for GUI development is Glade as discussed in Chapter 2.4.1.

6.2.2 Python GUI

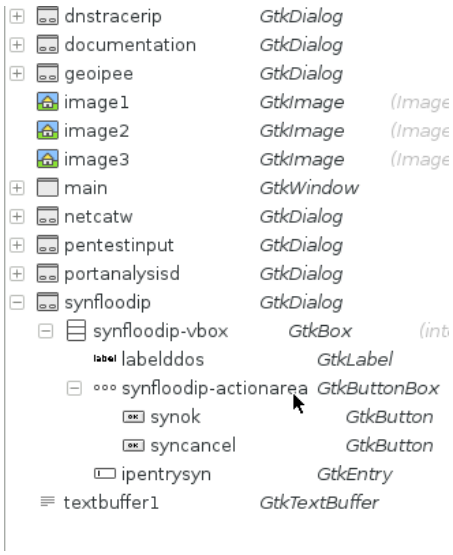


Figure 16: Sample Glade Elements

First order of business was setting up all the visual elements such as windows, buttons, images, text fields etc. in Glade as shown in figure 16. Using Glade makes it extremely easy to stay consistent with the design shown in section 5. Using good variable names and signal names is important as these will be used in the Python code to interact with the visual elements. Once everything is setup according to the design, it is time to code the event handlers in Python and import the generated Glade file.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
  <object class="GtkDialog" id="dnstracerip">
    <property name="width_request">300</property>
    <property name="height_request">140</property>
    <property name="can_focus">False</property>
    <property name="border_width">5</property>
    <property name="title" translatable="yes">Enter host name/IP</property>
    <property name="resizable">False</property>
    <property name="default_width">300</property>
    <property name="default_height">140</property>
    <property name="type_hint">dialog</property>
    <property name="has_resize_grip">False</property>
    <child internal-child="vbox">
      <object class="GtkBox" id="dialog-vbox10">
        <property name="can_focus">False</property>
        <property name="orientation">vertical</property>
        <property name="spacing">2</property>
        <child>
          <object class="GtkLabel" id="labeldnstracer">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
```

Figure 17: Sample Code Generated by Glade

Figure 17 shows the XML code that is generated by Glade; this will be imported into the main Python GUI code. The code generated is only for setting up tedious aspects of the GUI whereas the main functionality programmed in Python.

```
    def onButtonDnsOK(self, button):
        dialog = builder.get_object("dnstracerip")
        ipentry = builder.get_object("ipentrydns")
        ip = ipentry.get_text()
        subprocess.Popen(["xterm", "-e", "/root/dev/pengui/dnstracer.sh", ip])
        dialog.hide()

    def onButtonDnsCancel(self, button):
        dialog = builder.get_object("dnstracerip")
        dialog.hide()

    def onButtonFulltest(self, button):
        dialog = builder.get_object("pentestinput")
        dialog.show_all()

    def onButtonFullOK(self, button):
        dialog = builder.get_object("pentestinput")
        ipentry = builder.get_object("ipentryfull")
        ip = ipentry.get_text()
        subprocess.Popen(["xterm", "-e", "/root/dev/pengui/scripts/fulltest.sh", ip])
        dialog.hide()

    def onButtonFullCancel(self, button):
        dialog = builder.get_object("pentestinput")
        dialog.hide()

builder = Gtk.Builder()
builder.add_from_file("/root/dev/pengui/interface.glade")
builder.connect_signals(Handler())

window = builder.get_object("main")
window.show_all()

Gtk.main()
```

Figure 18: Sample code from interface.py

The Python code required importing the Gtk module which is the Python bindings for GTK and the subprocess module for launching sub-processes directly through Python applications. This will be important for launching the BASH scripts from the GUI without locking up the GUI process.

Figure 18 shows some sample code for the implementation of sub-processes within the Python GUI, this is the main interface file and is where the Glade elements are imported and utilised. User input can be received from Python text fields, stored as a variable and then used as a parameter for BASH scripts using the subprocess module.

Figure 19: User input pop-up window

To demonstrate the functionality of a random application using both the GUI and the BASH scripts, a sample run-through of 'NetCat' will be demonstrated.

Figure 19 is an example of how user input is received, if the user clicks on 'NetCat' through the main GUI then the pop-up box appears. 'NetCat' is an application similar to telnet but more powerful, it allows the user to make direct connections through a given port and perform a multitude of actions.

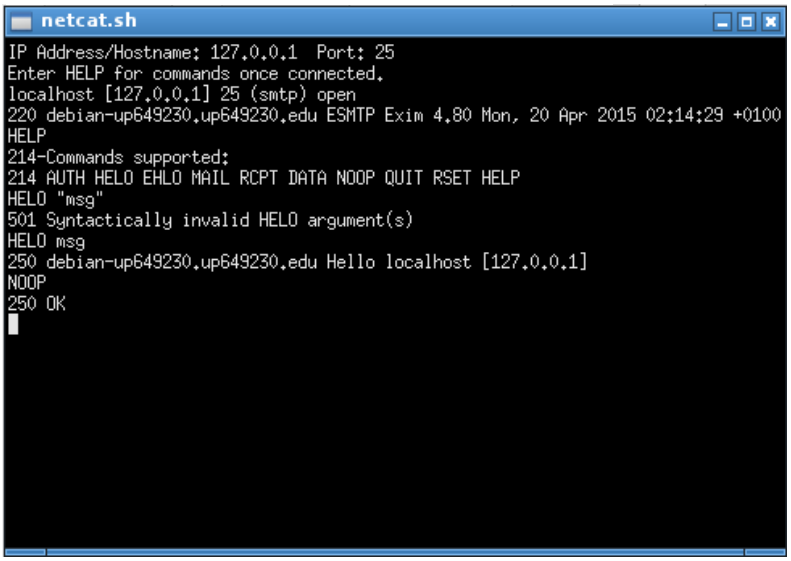


Figure 20: Sample 'NetCat' session

Once the user has input some data (using 127.0.0.1 & 25 as examples), the variables are passed onto the BASH script through the Python code as shown previously.

Figure 20 displays what the BASH script looks like once launched through a Python subprocess, the variables are used as parameters for 'NetCat' and the user can then operate the application very easily as shown.

Approriate error messages are returned to the user as shown in figure 21, 22 and 23. Valid inputs for this application is:

Figure 21: Invalid IP error

IP Addresses: 0-255.0-255.0-255.0-255

Figure 22: Invalid hostname error

Hostname: name OR name.domain OR www.name.domain

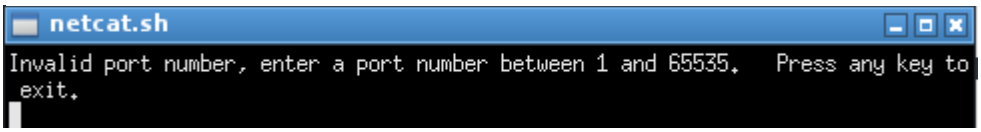


Figure 23: Invalid port error

Port: 1-65535

6.2.3 BASH Scripts

```
if [[ "$1" =~ ^([0-9]{1,3})[.]([0-9]{1,3})[.]([0-9]{1,3})[.]([0-9]{1,3})$ ]]
then
    for (( i=1; i<${#BASH_REMATCH[@]}; ++i ))
    do
      (( ${BASH_REMATCH[$i]} <= 255 )) || { echo "You have entered \
an invalid IP address, press any key to exit." >&2; read null; exit 1; }
    done

elif [[ ! "$1" =~ ^([a-zA-Z]{1,63})([.]{0,1})([a-zA-Z]{1,63})([.]{0,1})([a-zA-Z]{1,63})$ ]]
then
    echo "Not a valid IP address or Hostname. Press any key to exit"
    read null
    exit 1;
fi

if  [ "$2" -gt "65535" ] || [[ ! "$2" =~ ^([0-9]{1,5})$ ]]
then
    echo "Invalid port number, enter a port number between 1 and 65535.\
    Press any key to exit."
    read null
    exit 1;
fi

echo "IP Address/Hostname: $1\
    Port: $2"
echo "Enter HELP for commands once connected."
nc -b -v -v $1 $2
echo "Session terminated. Press any key to continue."
read null
exit 0;
```

Figure 24: BASH Script for NetCat application

The underlying code for the BASH script which is run for ‘NetCat’ is shown in figure 24. By evaluating the input through some simple regex pattern matching and if statements, the input data is verified or returns an error with information regarding the issue. If the variables are valid, they are passed onto the ‘NetCat’ application with appropriate options and parameters for verbose output to help guide the user. The options used for ‘NetCat’ in this script is ‘-b –v –v’ which means allowing broadcasts and double verbosity which is helpful for beginners, the parameters used is the IP Address/Hostname and Port number as received from the Python GUI.

```
#Intro sequence
echo "This is a full penetration testing script, please be patient as
this may take a while. The outputs of each individual test will
be displayed, you may scroll down after each part to assess the results.
Once you are satisfied, scroll down and press q.
Press ENTER to continue when ready or CTRL+C to quit."
read null
clear && echo "A full port scan is being performed with OS search, this may take
a while...."

#NMap port scan with OS search
/root/dev/pengui/scripts/nmap_port_digger.sh $1

# GeoIP of the host
clear && echo "We will now try to locate the country of origin for the hostname,
please press ENTER to continue."
read null
/root/dev/pengui/scripts/geoiplookup.sh $1

#Rootkit hunter
clear && echo "We will now perform two thorough rootkit scanners in case the
system has been infected through the network. Please press ENTER to
start, this may take a while...."
read null
rkhunter -c --vl

#Check Rootkit
chkrootkit

#DNStracer
clear && echo "We will now follow the chain of DNS servers to the source. Please
press ENTER to continue."
read null
/root/dev/pengui/scripts/dnstracer.sh $1
```

Figure 25: Full penetration test BASH script

The main reason for creating this artefact was to be able to automatically do penetration testing; the script shown in figure 25 is the script responsible for the full extensive test. It consists of a combination of many different BASH scripts much like the 'NetCat' script discussed above which can be run separately through the GUI, however the full script guides the user through every step of the process and runs the most important parts for the user to not to require prior knowledge.

6.2.4 Main Interface

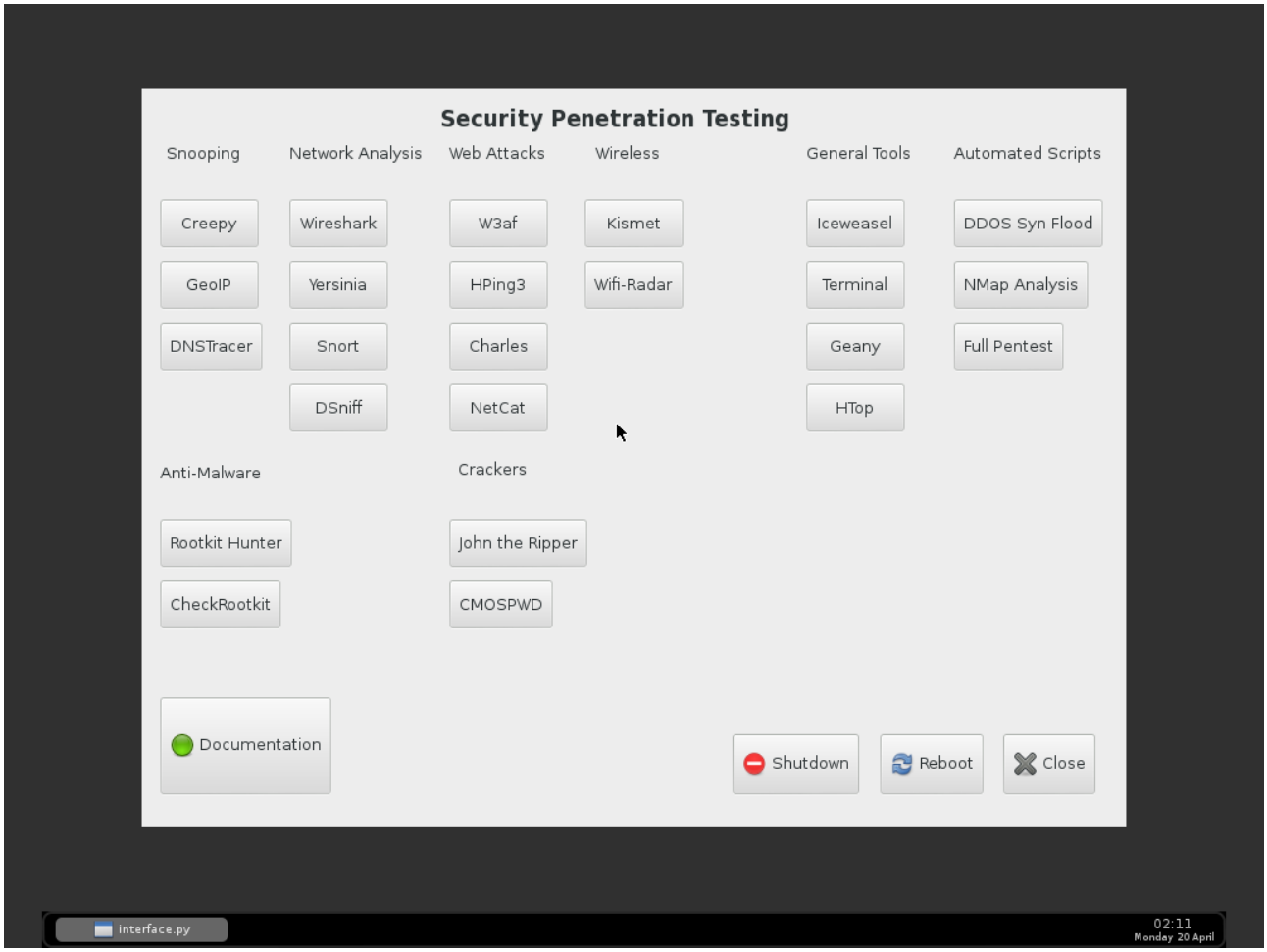


Figure 26: Desktop on boot-up

Figure 26 is what the user will see upon booting up the live distribution, the entire operating system runs on roughly 70MB RAM as shown in figure 27 due to the lightweight design which has been implemented.

Figure 27: Resource usage on boot-up

. The example shown in chapters 6.2.1 and 6.2.2 with the 'NetCat' application is simply one of the tools available through the GUI, there are many other applications and scripts available which are extremely powerful and would only be available to the advanced user without such a GUI.

The GTK+ theme 'Adwaita' is applied system-wide, this means that the Python GUI will have the same theme as any other GTK+ application launched throughout the distribution. For example, Iceweasel (the web browser) will have the same theme as the interface and all other GTK+ applications which allows for a more steamlined look.

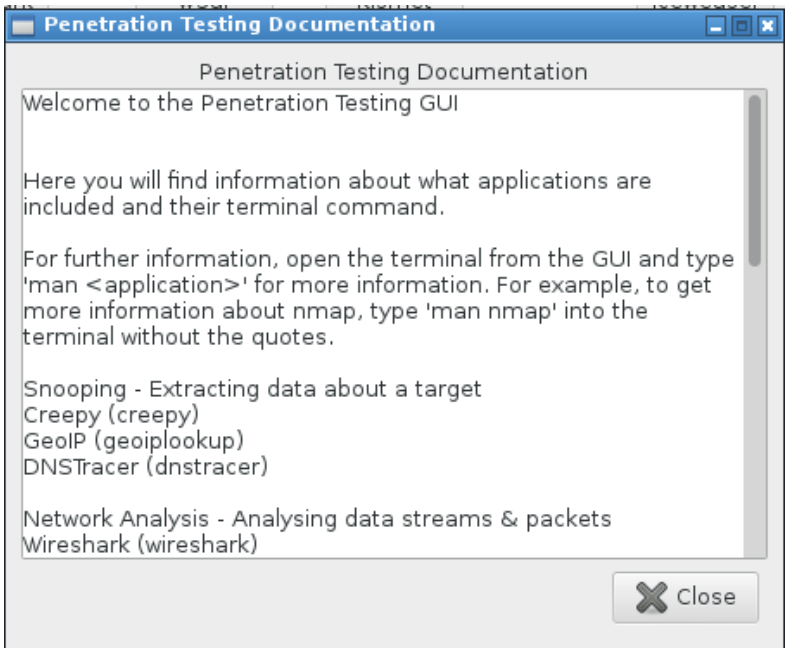


Figure 28: Implemented documentation window

Figure 28 is the documentation window which can be opened through the main interface. The idea is that the GUI should be so simple and guide the user so well that the documentation is not required, however if the user wishes to learn more about how each tool works the information is there and they are given the required information in order to be able to access the 'man' pages through the terminal.

7. Testing

The focus of this chapter is to lay out the different operations possible within the system, comparing the expected outcome with the actual output. Testing for several versions of the product will be made as the incremental build model has been followed which allows for constant re-designing, implementation and testing.

7.1 Validation & Verification

Table 1: Testing the Artefact

Ver.	Operation to be tested	Method of testing	Expected outcome	Actual outcome	Comments
V1.0	Booting up the distribution to the desktop environment	Booting up the computer from either a USB stick or CD, settings may have to be changed in BIOS to allow this	The operating system should boot up without errors in a timely manner, auto-connect to the network via Ethernet, load into the desktop environment and display the developed GUI	The operating system boots up without errors only if an Ethernet cable is plugged in due to DHCP assigning an IP. Everything else boots up without problems and the desktop environment is loaded with the Python GUI	Simply making sure the hardware is connected properly avoids all errors. A warning should be given to the user to ensure the Ethernet cable is connected before booting up.
V1.3 & V1.4	Checking whether or not less than 512MB RAM is used during the full testing script	Starting the full penetration testing script, checking the memory usage in htop during every phase	The RAM usage should be well under 512MB at peak, with some slack allowed due to different hardware requiring different drivers/modules	**V1.3:** 145MB~ RAM usage during regular usage. **V1.4:** More unnecessary modules are stripped and the RAM usage is 80-90MB during usage and 70MB on boot	For future versions, it would be feasible to have the operating system only require 128MB of RAM
V2.0	Running all applications and scripts from GUI	Clicking on every individual program button and seeing if it launches, sample input given for user input	Every application should launch without errors. If it is required for the user to give input, the parameter(s) should be forwarded properly and used to launch the program	Every single application launches and the appropriate user input is passed along to the terminal. Errors are given on a per-application basis	N/A

V2.2 & V2.3	Running scripts from GUI	Clicking on every script button, sample input given for user input	The chosen script uses the given input(s) as variable(s) for performing the penetration testing and outputs relevant data to the terminal.	Every script works as expected.	One script launches a DDOS attack on the given host. It is important that the user understand the potential dangers of this. Appropriate warning is given prior to running the script.
V2.4	Running full penetration testing script from GUI	Clicking on the full penetration testing script button, sample input given for user input	After the user has given the required inputs, the script introduces the user to the process and gives instructions on how to control the flow of the script. User should be given an opportunity to analyse the outputs at each stage and continue when they are ready.	The script works just as expected, however it takes roughly 10-15 minutes to run.	Some parts could be optimised or omitted to save time. However, as it is the full script another option would be to do a quicker scan version.
V2.4	Shutting down/rebooting gracefully	Shutting down/rebooting the system using the buttons on the GUI	The system should shut down/reboot immediately without any delay	The system shuts down/reboots instantly	N/A

7.2 Test Conclusion

Some of the tests were revisited on later iterations of the artefact due to the incremental build model; this showed that using that life cycle model was a great choice as fundamental aspects of the artefact (e.g. useless modules in kernel hogging up RAM space) would have gone un-altered using most other models such as waterfall. Overall, the testing phase was a great success as not only did it allow testing each aspect of the artefact, it also allowed for improvements after each iteration of the model and testing phase.

8. Planning

8.1 Initial Planning Phase

When starting to pursue this project, many aspects were unclear such as which methodology was going to be followed and other set-backs leading to not staying to the plan. Once set-backs occur, it's very easy to just ignore the plan set out in the beginning and simply continue at a leisurely pace. Figure 29 shows how the project was planned out in the beginning, it seemed good on paper at the time but didn't take methodology into consideration and it was especially bad because it also did not take other project into consideration.

ID	Task Name	Duration	Start	Finish
1	Aims and Objectives	2 days	Mon 03/11/14	Tue 04/11/14
2	Literature Review	7 days	Wed 05/11/14	Thu 13/11/14
3	Interview Client	1 day	Thu 13/11/14	Thu 13/11/14
4	Research & Choose Methodology	3 days	Fri 14/11/14	Tue 18/11/14
5	Analyse Requirements	1 day	Fri 14/11/14	Fri 14/11/14
6	Formalise Requirements	2 days	Mon 17/11/14	Tue 18/11/14
7	Use Lit Review To Make Design Choi	3 days	Fri 14/11/14	Tue 18/11/14
8	Use Case Diagrams	1 day	Wed 26/11/14	Wed 26/11/14
9	Design Distribution	3 days	Thu 27/11/14	Mon 01/12/14
10	Design GUI	3 days	Tue 02/12/14	Thu 04/12/14
11	Implement Distribution	3 wks	Fri 05/12/14	Thu 25/12/14
12	Implement GUI	4 wks	Mon 05/01/15	Fri 30/01/15
13	Implement BASH scripts	5 days	Mon 02/02/15	Fri 06/02/15
14	Convert Distribution To Live Distro	1 day	Mon 09/02/15	Mon 09/02/15
15	Compress Distribution	1 day	Tue 10/02/15	Tue 10/02/15
16	Test Product	5 days	Wed 11/02/15	Tue 17/02/15
17	Evaluate vs Requirements and Desig	1 wk	Wed 18/02/15	Tue 24/02/15
18	Conclusion Overview	2 days	Fri 20/02/15	Mon 23/02/15
19	Recommendations For Future Development	3 days	Tue 24/02/15	Thu 26/02/15
20	Personal Reflection	1 day	Fri 27/02/15	Fri 27/02/15
21	Final Thoughts	1 day	Sat 28/02/15	Sat 28/02/15

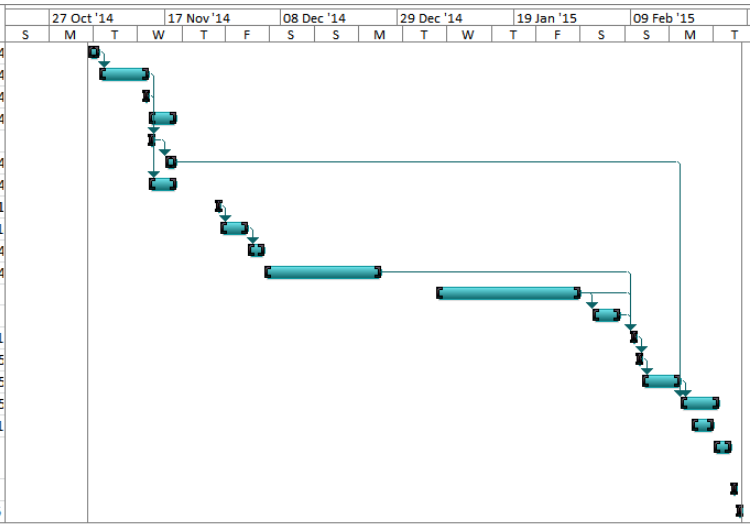


Figure 29: Initial Gantt Chart

Figure 30 shows the modifications made to the gantt chart in February, the reason for the big gap during the December-January is due to having other 2 other coursework projects needing handing in and it was not realistic to assume that any progress would be made during this time. When making the updated gantt chart, a methodology had already been chosen and adhered to and therefore timeframes could be given for updating the artefact by following the incremental build model. Version 1.0 of the artefact was the state it was at after the first cycle of the model, after which the other major versions were 1.5, 2.0 and 2.4, each time the requirements, design, implementation and testing was revised.

ID	Task Name	Duration
1	Aims and Objectives	2 days
2	Literature Review	7 days
3	Interview Client	1 day
4	Research & Choose Methodology	3 days
5	Requirements	3 days
6	Use Lit Review To Make Design Choi	3 days
7	Design Distribution	3 days
8	Design GUI	3 days
9	Implementation	1 wk
10	Convert Distribution To Live Distro	1 day
11	Compress Distribution	1 day
12	Test Product	5 days
13	Re-iteration: Version 1.5	1 wk
14	Re-iteration: Version 2.0	1 wk
15	Re-iteration: Version 2.4	1 wk
16	Evaluate vs Requirements and Desig	1 wk
17	Conclusion Overview	2 days
18	Recommendations For Future Development	3 days
19	Personal Reflection	1 day
20	Final Thoughts	1 day

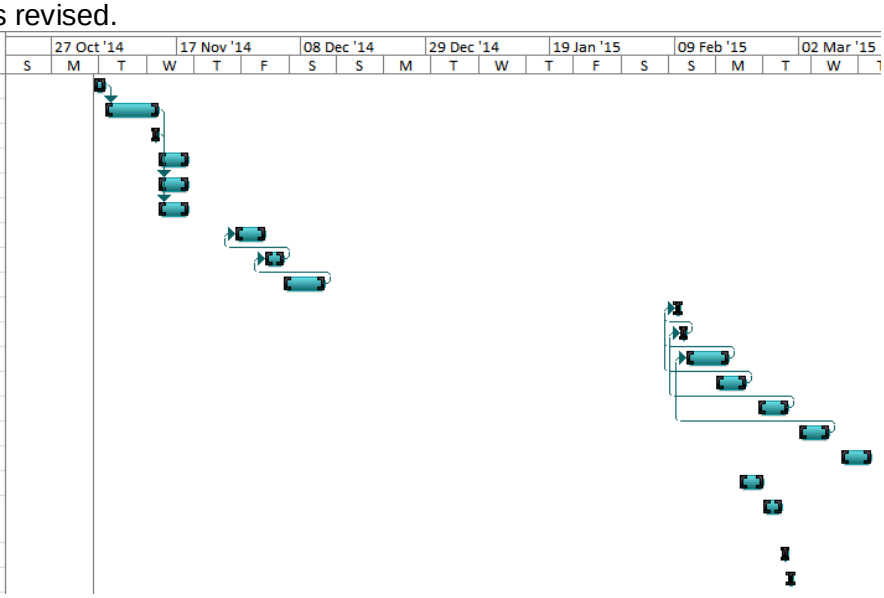


Figure 30: Updated Gantt Chart

9. Evaluation

The artefact will be evaluated in three different aspects:

- Evaluating the original aims & objectives to see how well thought out they were prior to researching the area and producing the artefact

- Assessing each requirement, judging whether or not the requirements were met and to what degree

- Evaluation of client feedback during the final interview, getting the clients take on how the most recent version of the artefact stacks up to the requirements given at the beginning and also after all of the versions

The reason for choosing these three aspects to focus on for the evaluation is that they are the basis and foundation of the project, they are the main things needed for being able to fully implement the artefact sans the design which needs to be re-modelled regularly.

9.1 Expectations and Comparisons

9.1.1 Aims & Objectives

Chapter 1.2 states that 'The aim of this project is to produce a lightweight, live Debian-based Linux distribution with a user-friendly GUI for the sole purpose of penetration testing.' Every single aspect of the aim has been achieved in the artefact. It is also stated in the aim that the distribution should be able to run off virtually any old hardware and whilst we have learned during the testing phase that the RAM and media requirements were fulfilled, the distribution simply hasn't been tested on more than a few machines. Although constraint has been mentioned in Chapter 1.4, it should still be noted that a main feature of the artefact is the fact that it should work on almost any old hardware so that aim has yet to be fulfilled and is not in the scope of the project at this current time.

Almost all objectives were satisfied fully at a basic level; however the documentation objective was not followed as intended initially. A lack of foresight led to the realisation that Linux has extremely good built-in documentation and as such, the user was guided to these 'man' pages for additional information. The documentation window was converted a brief summary of different categories and which applications belonged to which category including the terminal command for running them and reading the 'man' pages. Some basic information regarding the scripts was also included, however the scripts are automated an all instructions are shown on-screen which shows once again a lack of foresight and setting a realistic objective. After re-evaluating the objective for the reasons stated, all other objectives were met very well.

9.1.2 Requirements

Truncated versions of requirements are used in table 2, for the full versions refer to chapter 4.2.

Table 2: Evaluation of Requirements

Requirement	Evaluation	Priority
Booting without installation: Live distribution bootable off CD/USB	Largely satisfied requirement, the artefact boots without any user interaction straight into the operating system and immediately presents the user with the GUI. One minor issue is that the networking equipment (Ethernet cable, etc) needs to be plugged in prior to booting up the computer as an IP address needs to be allocated through DHCP.	10

Access to a wide variety of penetration testing software without configuration	This requirement can be considered moderately satisfied; much of this requirement was affected by the non-functional performance requirement. The main issue was that having a wide variety of applications to run from the GUI would make the interface very cluttered and difficult to navigate. Another issue is that there simply was not enough space to accommodate similar numbers of tools as other popular distributions such as Kali Linux which has hundreds of different tools as shown by Kali Linux Tools Listing (2015). Substitutions were made and in terms of functionality, not much is missing at all, there is simply less options to choose from. The last issue was not being able to include the penetration testing software MetaSploit due to the huge amount of hard drive space it required, however the functionality was replaced by several smaller applications.	10
All useful applications able to be launched from the GUI	Completely satisfied, every application necessary for penetration testing and every general purpose tool required is available with the click of a single button from the GUI. There is also some space left where 2 more rows (4 buttons per row) can be added for future versions.	10
Perform extensive automated penetration testing using BASH scripts	The main script for the artefact was a full extensive penetration testing script which takes roughly 20 minutes to complete, this may be and there may be a need for an update with a quicker versions of the test which is less extensive. It however works extremely well and all the user has to do is watch the results and continue when they have analysed the results. Great deal of effort went into making sure the user inputs were validated and guided the user to input the correct variables. There are also a few other miscellaneous scripts which all serve a different purpose.	8
Read documentation on how to use software from GUI	Documentation is one aspect of Linux which is amazing; this requirement was a bit short-sighted and was not that important as the GUI/scripts should be intuitive enough to not require documentation. Some basic documentation regarding the types of software and their respective commands was implemented, however the documentation provided is definitely not as extensive and in-depth as it should have.	3
Performance, entire artefact should require no more than 512MB RAM and fit on a 700MB CD	This requirement was more than just satisfied; the operating system has an idle RAM usage of less than 70MB and 100MB~ when running applications and scripts from the GUI. The total file size of the ISO file is 600MB~ which is perfect as it allows for future versions of the software to expand upon it for newer security threats.	10
Legal issues, all software included should be free and open-source. GPL license is ideal.	Completely satisfied requirement, all software is licensed under GPL or similar. There were some software packages which required a license or were not licensed in a manner which allowed for distribution but alternatives were found where needed.	10

Usability, non-security experts should easily be able to perform penetration testing	Another completely satisfied requirement, it would be hard to make the artefact any more user friendly than it already is. The user only needs to understand some basic computer skills and everything else is guided by scripts. If any user input is required, it is described in great detail. However, there is room for more scripts for the more advanced software for future work.	7

9.1.3 Client Feedback & Evaluation

A fairly long discussion took place after the latest (final in the scope of this project) version of the artefact was sent to the client for testing. The testing process was very simple as the distribution runs live, it's simply a case of the client downloading it and booting it up.

An idea given by the client was to have a separate version which is more feature-rich aimed towards users who have the resources to install the distribution on a dedicated computer; this would go well as an alternative to this lightweight live version.

Overall, the client was mostly happy with the final version given as it satisfied all the requirements after a lengthy development process using the incremental build model where the requirements were always evolving based on experiences with the artefact.

10. Conclusion

10.1 Artefact Development

The artefact was for the most part a success; much of the success can be attributed to adherence to the incremental build model along with the initial interview with the client and the follow-up interviews. Being able to get the feedback from the client after each iteration of the life cycle, it was easy to adjust the requirements to suit the needs of the client but also to make sure they were manageable.

Objectives set at the beginning were predominantly followed accordingly, however not much kernel modification was performed past removing unnecessary modules and bloat. Some objectives were however not followed in the correct order as this project can be considered two sub-projects, as such; many elements of the Python GUI and the distribution itself were developed in parallel.

One major issue was however the requirement for documentation within the Python GUI, the main reason this requirement was not fulfilled properly was because of its low priority level and Linux applications in general already having extremely good documentation though the ‘man’ command. The documentation was replaced with a brief summary of the Python GUI and its capabilities, it may still be very useful for the client, however it does not go in-depth about the inner workings of each application.

10.2 Personal Reflection

There was quite a lot of prior knowledge of Linux as a hobbyist; the initial spark for the project was an effort to combine the programming knowledge gained from time spent at University with a passion for operating systems. This type of artefact is very unorthodox for an engineering project, there was some difficulty adhering to software engineering conventions but in the end the finished product is something to be proud of.

As with most projects, failing to plan is planning to fail. Even with a lot of slack time, not scheduling tasks over Christmas etc. there were still issues adhering to the plan 100%. The main issue with the plan was not considering other units and how they would affect the time spent on the project, giving the planning phase more thought would have avoided many headaches and frustration.

10.3 Recommendations for Future Development

Here are some examples of possible improvements that could be made in the future:

- The number of tools/applications available in this artefact is much lesser than those of other penetration testing distributions; this is largely due to the fact that it is a requirement that it is under 700MB in order for it to fit on a CD. One way of improving this would be to use more advanced compression techniques in order to allow for more content in those 700MB. When doing this, it would be a good idea to create a table for all the applications/scripts included and comparing the file size with the importance of the applications in order to objectively rank the priority level of each aspect of the distribution.

- Outputting the results gathered by the scripts to a temporary file and having the option of saving it to an external medium or e-mailing it to a person. One way of doing this would be by using the SMTP protocol and auto-mounting external devices such as USB flash drives. This could be a great idea because the person running the scripts might not be the most technically literate when it comes to security so the data could be passed onto a more qualified professional.

- Testing and implementing compatibility on a much larger scale, as of now the artefact is only confirmed to run properly on the developers machine and the clients, additional drivers/modules and settings will very likely need to be implemented in order to improve

compatibility with many different kinds of systems, especially older ones as the artefact is aimed towards slower systems. Releasing it to the public and getting bug reports is an excellent real-life implementation of this idea.

- Developing a system which requires absolutely no user input, you simply put in the CD/USB and boot up the computer. A series of scripts would take over the testing and at no point does the user need to give any input. This type of system is possible but would be much less powerful at the cost of user-friendliness.

10.4 Final Thoughts

Much appreciation for the project supervisor allowing such an unorthodox project to be pursued, at its current state the artefact was a success and could be released for public use, or at least as an early alpha testing version. The artefact will not be abandoned, there has always been a will to develop such an artefact but not enough incentive or drive to pursue it, after seeing what is possible in only a few months the artefact will be developed further in the coming months and be released to the public as a free open-source distribution. By doing this, the compatibility issues will be much lesser as more people can test it and send bug reports.

11. References

Muniz, J., Aamir, L. (2013). *Web Penetration Testing with Kali Linux.* Birmingham: Packt Publishing Ltd.

Offensive Security. (2015). *Kali Linux Official Documentation.* Retrieved from: http://docs.kali.org

Nemeth, E., Snyder, G., Hein, T. R. (2006). *Linux Administration Handbook* (2nd ed). Massachusetts: Pearson Education, Inc.

Krause, A. (2007). *Foundations of GTK+ Development.* New York: Springer-Verlag

McLean, G. (2005). Building GTK Apps with Glade. *The Perl Review*, 1(3), 6-10.

Neckel, T. (2015). *Scripting with BASH and Python.* [Lecture Notes]. Retrieved from: http://www5.in.tum.de/lehre/vorlesungen/progcourse/bashpython/bash-2x4.pdf

About Geany. (2010). Retrieved from the Geany website: http://www.geany.org/Main/About

Kali Linux Tools Listing. (2015). Retrieved from the Kali Linux website: http://tools.kali.org/tools-listing

Cappos, K., Samuel, J., Baker, S., Hartman, H. J. (2008). *A Look In The Mirror: Attacks on Package Mangers.* Retrieved from: http://www.cs.arizona.edu/people/jsamuel/papers/TR08-02.pdf

Kostaras, N., Xenos, M. (2010). A study on how usability flaws in GUI design increase mouse movements and consequently may affect users' health. *Behaviour & Information Technology,* 30(3), 425-436. http://dx.doi.org/10.1080/0144929X.2010.529943

Beaudouin-Lafon, M., Mackay, W. (n.d.) *Prototyping Tools and Techniques.* Retrieved from: https://www.kth.se/social/upload/52ef5ee4f2765445a466a28a/mackay-lafon-prototypes-52-HCI.pdf

Dalheimer, M., K. (2002). *Programming with Qt (2nd edition)*. Balthasarstr: O'Reilly Verlag GmbH & Co.

Decrem, B. (2004). Desktop Linux: Where Art Thou?. *Queue – Open Source Grows Up,* 2(3), 48-56.

Yu, L., Ramaswamy, S., Bush, J. (2008). Symbiosis and Software Evolvability. *IT Professional*, 10(4), 56-62.

Larman, C., Basili, R., V. (2003). Iterative and Incremental Development: A Brief History. *Computer*, 36(6), 47-56. http://doi.ieeecomputersociety.org/10.1109/MC.2003.1204375

12. Appendices

12.1 Appendix A - PID

Project Initiation Document

Basic details

Student name:	Josef Abbasikadijani
Draft project title:	Debian-based live Linux Distribution for Penetration Testing
Course:	Computer Science
Client organisation:	
Client contact name:	
Project supervisor:	Benjamin Aziz

Outline of the project environment and problem to be solved

The client for this project is my brother, he requires a Linux distribution for penetration testing but the current solutions are too bloated and many require installation for full functionality. I propose a Debian-based Linux distribution which can be run from a CD or USB which is very lightweight and has a whole new desktop GUI developed for making the penetration testing process a lot easier. The distribution will also come with an assortment of BASH scripts aimed to make the process automated for the user.

Project aim and objectives

The aim of this project is to create a Debian-based live Linux distribution and develop a GUI to make the penetration testing process user-friendly.

Objectives for this project are:

- Research general Linux software, penetration testing software packages, existing similar distributions and libraries/programming language for the GUI
- Setup a minimalist Debian installation
- Configure the distribution and change/add software packages as needed
- Compress and convert distribution to a live distribution
- Design and develop the GUI along with scripts
- Test the entire project
- Evaluate it

Project deliverables

The deliverable artefacts are the distribution and the GUI. Together, they make a complete package which the client is able to use to easily perform penetration testing.

The deliverable document included will be a report detailing the background of the artefacts, the research that went into it, the designing, implementation, testing and evaluation amongst other things. Every step of the process will be analysed and justified, with insight added on current methods and improvements that can be made.

Project constraints

The constraints are that it is a very ambitious project; there will not be enough time to implement enough features for it to be released to the public. Compatibility will also be an issue, being able to run it on any hardware is important but for now making it work for the client is the biggest priority.

Project approach

The first step will be to simply research most existing popular penetration testing distributions, learning their advantages and disadvantages. After this, I will interview the client to learn what specific requirements are needed and then analyse the data to set some formal requirements. I will also need to research different programming languages and libraries available for developing the GUI. After this I will use an incremental build lifecycle and keep the client involved in the process until the product meets all the requirements.

Facilities and resources

Due to the nature of the project, all the hardware necessary will be self-funded. The bulk of the artefacts will be produced at home on laptops and PCs; one problem is compatibility as I will not have enough computers to test it on. The live distribution will be tested off a USB drive, but it will be developed within a VM. The software used to produce the artefacts are countless and not in the scope of a PID, but the general text editors, programming IDEs and such will be required but will be of no issue as they are all free and open-source under the GPL.

Log of risks

There is always a risk of data loss; regular backups (perhaps automated?) must be performed. These backups can be on a USB stick, but there also needs to be off-site backups in the form of some cloud storage or FTP server. Unforeseen events could also get me off track, illness or not being able to work for a period of time could get me behind schedule. It is important to allow for some slack.

Risk	**Prevention**
Loss of data	Regular off-site backups
Illness	Slack time
Errors in distribution fundamentals	Make sure of the incremental build model, revert to a previous distribution model
Bugs	Extensive testing phase, communication with the client

Starting point for research.

First off, extensive research has to be made into what current distributions are available that serve the same purpose, specifically into their philosophies and software included in the distribution. I will then have to research what programming language and libraries are most appropriate for developing the GUI.

Breakdown of tasks

What do you need to do to create the artefact and write the report? Walk through your proposed approach and list the tasks.

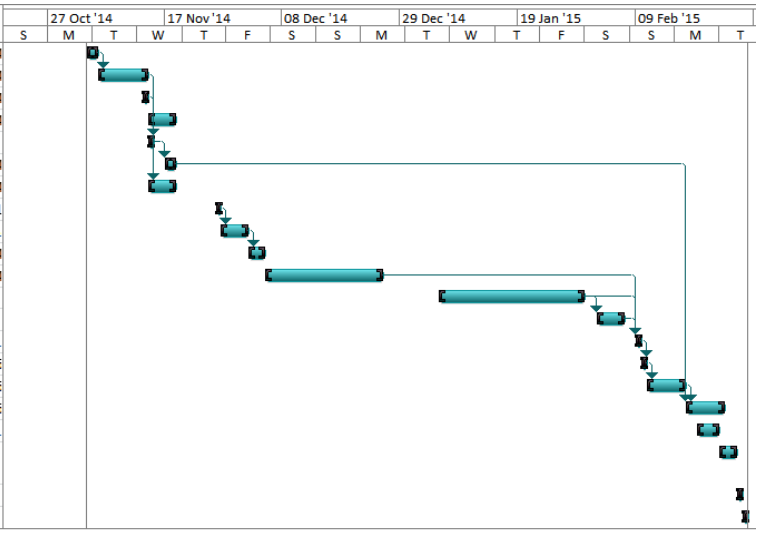
- Write aims and objectives
- Do an extensive lit review on existing distributions and possible programming languages and libraries for the GUI
- Interview the client to get requirements
- Research different methodologies and life cycles and choose an appropriate one
- Analyse the requirements given by client
- Create a formal list of functional and non-functional requirements based on the data given by the client and my analysis
- Start the design process, using information gathered in lit review
- Design the distribution and the GUI individually
- Implement the distribution, due to the complexity this might take a few weeks
- Implement the GUI, programming this along with the BASH scripts could take a month
- Convert the Linux distribution to a live one
- Compress it to make it under 700MB
- Test the product
- Evaluate the product against the requirements and the design
- Conclude the project, giving information on future development and personal reflections

Project plan

What are you going to do when? (This may be an attached output from MS Project etc.)

What risks to the success of the project have you identified? What steps can you take to minimise them? Note that plans can change over the course of the project, so this plan should be maintained.

ID	Task Name	Duration	Start	Finish
1	Aims and Objectives	2 days	Mon 03/11/1	Tue 04/11/14
2	Literature Review	7 days	Wed 05/11/1	Thu 13/11/14
3	Interview Client	1 day	Thu 13/11/14	Thu 13/11/14
4	Research & Choose Methodology	3 days	Fri 14/11/14	Tue 18/11/14
5	Analyse Requirements	1 day	Fri 14/11/14	Fri 14/11/14
6	Formalise Requirements	2 days	Mon 17/11/1	Tue 18/11/14
7	Use Lit Review To Make Design Choi	3 days	Fri 14/11/14	Tue 18/11/14
8	Use Case Diagrams	1 day	Wed 26/11/1	Wed 26/11/1
9	Design Distribution	3 days	Thu 27/11/14	Mon 01/12/1
10	Design GUI	3 days	Tue 02/12/14	Thu 04/12/14
11	Implement Distribution	3 wks	Fri 05/12/14	Thu 25/12/14
12	Implement GUI	4 wks	Mon 05/01/1	Fri 30/01/15
13	Implement BASH scripts	5 days	Mon 02/02/1	Fri 06/02/15
14	Convert Distribution To Live Distro	1 day	Mon 09/02/1	Mon 09/02/1
15	Compress Distribution	1 day	Tue 10/02/15	Tue 10/02/15
16	Test Product	5 days	Wed 11/02/1	Tue 17/02/15
17	Evaluate vs Requirements and Desig	1 wk	Wed 18/02/1	Tue 24/02/15
18	Conclusion Overview	2 days	Fri 20/02/15	Mon 23/02/1
19	Recommendations For Future Development	3 days	Tue 24/02/15	Thu 26/02/15
20	Personal Reflection	1 day	Fri 27/02/15	Fri 27/02/15
21	Final Thoughts	1 day	Sat 28/02/15	Sat 28/02/15



Legal, ethical, professional, social issues

What are the legal/ethical/professional/social issues that may impose constraints on the project? How will you ensure that they will be complied with, or what steps will you take to avoid/mitigate their effects?

What research ethics approval (if any) is needed for your work? Have you completed an ethical examination checklist and consent form if necessary? **Remember** – this is obligatory.

Signatures

	Signature:	Date:
Student		
Client		
Project supervisor		

12.2 Appendix B – Ethical Checklist

PJE40 and PJS40

University of Portsmouth

Ethical Examination

Undergraduate Final Year Projects

School of Computing

Faculty of Technology

This document describes the issues that you need to consider before you start your investigations. This is particularly important where your work may involve other people (human subjects) for the collection of information as part of your project work.

The examination takes the form of a checklist of 12 questions. Each question has come guidance notes.

Consider each question is turn and check the box for Yes or No.

You are then asked to write a short entry explaining the reason for your reply.

For example:

6. Are you in a position of authority or influence over any of the human subjects in your study? Comments: ***Although all the human subjects will be staff members at the University of Portsmouth, none of them are in my own department or***	Yes ☐ No ☒

area and none are subordinate to me in a management structure. Therefore I can see no way that I could have undue influence over them They could take part completely voluntarily.	

If a grey box is ticked then your project ideas need to be looked at more closely, and you MUST discuss this matter with your project Supervisor.

The final sections deal with Information Sheet(s) and Informed Consent, and you must attach any extra documents concerning these (where relevant) to this Ethical Examination at time of the submission of your Initial Report.

Ethics Information: 12-point Checklist

1. Will the human subjects be exposed to any risks greater than those encountered in their normal lifestyle? ***For example: could the study induce psychological stress or anxiety; is more than mild discomfort or pain likely to result from the study; will the study involve prolonged or repetitive activities?*** ***Investigators have a responsibility to protect human subjects from physical and mental harm during the investigation. The risk of harm must be deemed to be no greater than in their normal lifestyles.*** Comments:	Yes No ☐ ☒
2. Will the human subjects be exposed to any non-standard hardware or non-validated instruments? *Human subjects should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, or typical interactions with desktop, laptop PC's, tablet PC's, PDA's or mobile phones are considered non-standard (for example, using a VR room) nor should they be subjected to non-validated instruments e.g. unscrutinised questionnaires.* Comments:	Yes No ☐ ☒
3. Will the human subjects voluntarily give consent? ***If the results of an evaluation (for example) are likely to be used beyond the term of the project (for example, software is to be deployed or data is to be published), then signed consent is necessary. A separate consent form should be signed by each human subject. Return of a consent email can constitute written consent if this has been made clear to the human subject.*** ***Otherwise verbal consent is sufficient and should be explicitly requested in the introductory script (Information Sheet).***	Yes No ☒ ☐

Comments:	
4. Will any financial, or other, inducements (other than reasonable expenses and compensation for time) be offered to human subjects? *The payment of human subjects must not be used to coerce them against their better judgement, or to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.* Comments:	Yes No ☐ ☒
5. Does the study involve human subjects who are unable to give informed consent (for example: children under 18, people with learning disabilities, unconscious patients). *Parental consent is required for human subjects under the age of 18. Additional consent is required for human subjects with impairments, and people assessed to be lacking in mental capacity. If consent is gained from a person other than the human subject themselves e.g. a parent, then written consent* ***must*** *be obtained.* Comments:	Yes No ☐ ☒
6. Are you in a position of authority or influence over any of your human subjects? ***A person in a position of authority or influence over any human subject must not be allowed to pressurize them to take part in, or remain in, any study.*** Comments:	Yes No ☐ ☒

7. Are the human subjects being provided with sufficient details of the study at an appropriate level of understanding? ***All human subjects should be able to understand the information provided in any documentation and/or verbal information they receive about the experiment or study. They have the right to withdraw at any time during the investigation, and they must be able to contact the investigator after the investigation. They should be given the details of both student and supervisor as part of the debriefing. This information should be in the introductory script (Information Sheet).*** Comments:	Yes No x ☐
8. After the study, will human subjects be provided with feedback about their involvement and be able to ask any questions they may have about this involvement? *If the human subjects request further information, the investigator must provide the human subjects with sufficient details to enable them to understand the nature of the investigation and their part in it.* Comments:	Yes No x ☐
9. Will the human subjects be informed of the true aims and objectives of the study? *Withholding information or misleading human subjects is unacceptable if human subjects are likely to object or show unease when debriefed. It must be clear to human subjects if information is being withheld in order to elicit a true response. This should precede any analysis of the data.* Comments:	Yes No x ☐

10. Will the data collected from the human subjects be made available to others (where appropriate and only in relation to this research study), and be stored, in an anonymous form? *All human subject data (hard-copy and soft-copy) should both be stored securely and, if appropriate made available, in an anonymous form. Making human subject data available to a third party may be relevant where a student is taking part in a wider research project eg. for a member of the University staff, in which case anonymity of human subject data must be preserved.* Comments:	Yes [x] No [ ]
11. Will the study involve NHS patients, staff, or premises? ***If yes, then an application must be made to the appropriate external NHS Local Research Ethics Committee (LREC). For projects other than postgraduate research studies, the length of time for gaining external approval may not fit into a project timescale.*** Comments:	Yes [ ] No [x]
12. Will the study involve the investigator and/or any human subject, in activities that could be considered contentious, morally unacceptable, or illegal? *If yes, then further approval must be sought. For example: a project involving the study of pornography on the web will fall into this category. It is possible that the project may not be allowed to proceed.* Comments:	Yes [ ] No [x]

Please attach the following:

- Any Information Sheet(s) or introductory script(s) that the investigator has created for the benefit of the human subjects in the study. ***(See http://www.btinternet.com/~trking for examples of Information Sheets that set out details of a research study for human subjects).***

- Any documentation that the investigator has created to gather informed consent from the human subjects. This may be an Informed Consent Form, or a form of wording used to get verbal consent. ***(See http://www.btinternet.com/~trking/icf.htm for an example of an Informed Consent Form for research study with human subjects).***

==========

By signing this form, I AGREE to abide by the decisions made in the above points.

If at any time during my project, my answers would change from a white box to a grey box, then I MUST seek re-approval for my project. I understand that if I do not do so, then it is possible that I may FAIL the project component of my course.

Student name: ……..…… Jupiter number: ……

Student signature: …….. Date ……

==========

Supervisor signature: ……. Date ……..

University of Portsmouth

Certificate of Ethics Review

Project Title:	Debian-based Live Linux Distribution for Penetration Testing
User ID:	649230
Name:	Josef Abbasikadijani
Application Date:	02/04/2015 02:15:33

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

SchoolOrDepartment: SOC
PrimaryRole: UndergraduateStudent
SupervisorName: Benjamin Aziz
HumanParticipants: No
PhysicalEcologicalDamage: No
HistoricalOrCulturalDamage: No
HarmToAnimal: No
HarmfulToThirdParties: No
OutputsPotentiallyAdaptedAndMisused: No
Confirmation-ConsideredDataUse: Confirmed
Confirmation-ConsideredImpactAndMitigationOfPontentialMisuse: Confirmed
Confirmation-ActingEthicallyAndHonestly: Confirmed

Certificate Code: 9D07-179C-41F4-3589-00D4-6018-00B7-1629 Page 1