

Custom Secure Linux Distribution Generator

Humaid bin Mohammed ALQASSIMI

BSc (Hons) Computer Science

Final Year Dissertation

Supervised by Prof. MANUEL MAAREK

April 2021

The copyright in this dissertation is owned by the author. Any quotation from the dissertation or use of any of the information contained in it must acknowledge it as the source of the quotation or information.

Declaration

I, Humaid bin Mohammed ALQASSIMI confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Date: April 21, 2021

Signed: Humaid AlQassimi

Abstract

This project aims to provide a web application which allows you to build your own custom operating system (Linux distribution) with security settings which matches your requirements, abstracting complicated Linux configurations to a simple wizard which adapts to user's knowledge level. The final result is an installable system disc/image, which may be placed on a USB flash drive to install systems or used without an install (live mode).

This allows anyone, including individuals, organisations, or enthusiasts to create an operating system that matches their use case and security requirements, without being security experts. Security experts are also able to use the tool to build a safer operating system.

Table of Contents

Declaration	i
Abstract	ii
Table of Contents	iv
List of Figures	v
List of Tables	vi
List of Codes	vii
1 Introduction	1
1.1 Motivation	1
1.2 Aims and Objectives	2
2 Background	4
2.1 Linux Distributions	4
2.1.1 What is Linux & Distributions?	4
2.1.2 Standard Libraries	4
2.1.3 Init system	5
2.2 Specific-Purpose Linux Distributions	8
2.3 Linux Surveys	9
2.4 Hardening Linux	10
2.5 OS Customisation & Provisioning	12
2.6 OpenBSD & Proactive Security	14
2.7 Usable Security	15
2.8 Conclusion	17
3 Prototype Development & User Interface	18
3.1 Overview	18
3.2 Requirement Analysis	19
3.2.1 Priority Scheme	19
3.2.2 Functional Requirements	19
3.2.3 Non-functional Requirements	19
3.2.4 Requirements and Objectives	19
3.3 The Wizard	20
3.3.1 Questions, Options & Effects	20
3.3.2 Security Stringency Score	23

4	Web Implementation	27
4.1	Overview	27
4.2	Choosing Appropriate Tools	27
4.3	Development Environment	27
4.4	Architecture	28
4.4.1	Interaction between web and builder components	28
4.5	Web Interface	29
4.5.1	Packages Listing	29
5	Builder System Implementation	32
5.1	Overview	32
5.2	What is a Distribution	32
5.3	Our Base	33
5.3.1	Legal Considerations	33
5.4	Builder Process & Tools	35
5.4.1	Extraction Process	35
5.4.2	Customisation Process	36
5.4.3	Building Process	40
6	Evaluation	42
6.1	Evaluation Strategy	42
6.1.1	Technical Evaluation	42
6.1.2	Usability Evaluation	44
6.2	Evaluation Results	44
6.2.1	Usability Results	44
7	Conclusion	47
7.1	Overview	47
7.2	Limitations & Future Work	47
7.2.1	More Customisability	47
7.2.2	Better Usability & User Experience	48
7.2.3	Further Improvements in Usable Security	48
7.2.4	Human Factor	48
7.2.5	Security Interface	48
	References	50
	Appendix A Usability Survey	54
	Appendix B Usability Survey Results	64

List of Figures

2.1	Comparison between musl and other C/POSIX standard library implementation for Linux [5].	5
2.2	systemd Architecture	7
2.3	Screenshot of distrochooser	9
2.4	OpenBSD 5.3 Release Cover	15
2.5	Password management techniques survey	16
3.1	Prototype wizard screenshot	20
4.1	Architecture Diagram	29
4.2	Use Case Diagram	30
4.3	Packages selection on the web interface.	31
4.4	Packages categorisation on the web interface.	31
5.1	A screenshot of a Lubuntu system.	34
6.1	Gender of participants	45
6.2	Primary Operating System of participants	45
6.3	Correlation with Linux/Unix confidence and consideration of using the system	46

List of Tables

3.1	MoSCoW Priority Scheme explained [37].	19
3.3	Functional Requirements.	24
3.4	Non-functional Requirements.	25
3.5	Relationship between the objectives and requirements.	25
3.6	Security measurements depending on the security stringency score.	26

List of Codes

2.1	A system configuration defined in the Nix expression language	13
5.1	Example of customisation with proot	38
5.2	Example of customisation with chroot	38
5.3	Snippet from the beginning of the customisation script	39
5.4	Snippet from the end of the customisation script	40

Chapter 1 Introduction

Computers became an essential part of our lives, we use them for everything from personal work to business. We increasingly store more sensitive data on our computers, and this raises an issue about the integrity and confidentiality of the data.

An operating system's only task isn't to run programs and manage the different aspects of the system, its objective is also to prevent unauthorised parties from accessing your information, or gaining access of your system.

A major issue with common operating systems, such as Apple's macOS and Microsoft's Windows is that these operating systems are proprietary. The issue with proprietary software is the inability to audit the source code¹. Often these proprietary operating systems require a license to run (such as the case with Windows), or may only run on a specific set of hardware (which is the case with macOS).

The need for alternatives operating systems is growing in the past few years, with the most popular being Linux. Linux is already used in critical servers, but it hasn't been widely adopted for desktop use cases. We use Linux to refer to any Linux-based operating system (or we may call it a distribution), as Linux by itself is just a kernel (which is the core of an operating system).

1.1. Motivation

Creating a customised Linux distribution requires deep understanding of Linux and its ecosystem, with information and tutorials online regarding creating a custom Linux distribution are usually out of date or sparse. Previously, there was a platform called SUSE Studio which allowed users to create custom builds of OpenSUSE (SUSE's Linux distribution). Unfortunately, this was taken down² on February 2018 and there are no alternatives.

This makes it difficult for system administrators to create a custom Linux image to mass

¹Although some software components in macOS and Windows are fully open-source or source-available through their initiative, the major aspects of the system isn't available for public – which is the main issue here.

²Due to a corporate decision, it was replaced by another commercial product.

deploy on systems, or for hobbyists to create a system which matches their preferences. Use cases for a system like this is vast, which we cannot easily summarise. There are tools available that allow users to remaster Linux distributions, but they are either too obscure and technical or outdated.

Furthermore, configuring Linux systems to be secure is a complicated task and involves different applications, kernel configurations, and recommendations to be applied. This makes the task of securing Linux systems a particularly broad scope.

1.2. Aims and Objectives

The aim of this project is to allow users to easily create a custom Linux distribution, with the goal of usable security. This is done through a web application, which includes a wizard to collect the user's specifications. This is used to automatically determine how to set up the distribution. It would also have an advanced settings page which allows users with Linux system experience further customise the system.

This web application allows the users to go through the entire process of creating their custom Linux distribution without having to install packages or software on their system.

The generated Linux distribution should have security preferences which matches the user's needs, and should allow users to make well-informed decisions when selecting security policies. There should also be a way to allow users to verify the integrity of the generated distribution.

The main objectives of this project can be summarised as the following (each includes a short identifier in bold):

- Align security policies and settings in the Linux distribution, based on various security guidelines (**Policies**).
- Provide usable security in Linux systems (**Usable Security**).
- Allow users to make informed decisions regarding security policies (**Informed Policies**).
- Create a wizard which collects user requirements for the Linux distribution (**Wizard**).
- Implement a server that tailors a Linux distribution based on a given specifications (**Builder**).

- Provide a tool or mechanism which allows users to trust the integrity of the final image (**Integrity**).

Chapter 2 Background

The research topic is about creating a system which generates a secure Linux distribution, with a focus on creating usable security. We'll look into existing solutions on creating Linux distributions, and the difference between them.

We'll also explore different concepts and research key subjects which are relevant to this project.

2.1. Linux Distributions

2.1.1. What is Linux & Distributions?

When we refer to Windows or macOS, we consider them directly as operating systems. Systems like Windows, macOS, or even OpenBSD includes the user-space applications bundled with the kernel. This is the main reason why you can find an official OpenBSD or Windows installation image, but with Linux you won't find an installation file on their official websites [1].

Generally, an operating system consists of a kernel – the core of the operating system, which handles memory and process management, hardware and device management, and provides a standard interface to applications (user-space programs). We also have the user-space applications and libraries, which are really important, because without them we won't have a usable system [2].

2.1.2. Standard Libraries

Operating systems require an implementation of the standard C library, which is really important because the vast majority application has calls to the standard C library as defined by the Portable Operating System Interface (POSIX) standards. POSIX is defined by the IEEE¹ Computer Society [3].

Linux uses the GNU standard library (`glibc`), which is most commonly used in Linux

¹IEEE stands for the Institute of Electrical and Electronics Engineers.

distributions. Alternatives of the standard library exists, such as `musl` – a more efficient and lightweight implementation of the standard C library. `musl`'s main focus is on correctness and safety, which is a main focus with our project. Unfortunately, we will not be using it due to the huge complexities that we'll have to undertake to swap the standard C library on a pre-existing system – which is out of the scope of this research project [4].

Bloat comparison	musl	uClibc	dietlibc	glibc
Complete .a set	426k	500k	120k	2.0M †
Complete .so set	527k	560k	185k	7.9M †
Smallest static C program	1.8k	5k	0.2k	662k
Static hello (using printf)	13k	70k	6k	662k
Dynamic overhead (min. dirty)	20k	40k	40k	48k
Static overhead (min. dirty)	8k	12k	8k	28k
Static stdio overhead (min. dirty)	8k	24k	16k	36k
Configurable featureset	no	yes	minimal	minimal
Behavior on resource exhaustion	musl	uClibc	dietlibc	glibc
Thread-local storage	reports failure	aborts	n/a	aborts
SIGEV_THREAD timers	no failure	n/a	n/a	lost overruns
pthread_cancel	no failure	aborts	n/a	aborts
regcomp and regexec	reports failure	crashes	reports failure	crashes
fnmatch	no failure	unknown	no failure	reports failure
printf family	no failure	no failure	no failure	reports failure
strtol family	no failure	no failure	no failure	no failure
Performance comparison	musl	uClibc	dietlibc	glibc
Tiny allocation & free	0.005	0.004	0.013	0.002
Big allocation & free	0.027	0.018	0.023	0.016
Allocation contention, local	0.048	0.134	0.393	0.041
Allocation contention, shared	0.050	0.132	0.394	0.062
Zero-fill (memset)	0.023	0.048	0.055	0.012
String length (strlen)	0.081	0.098	0.161	0.048
Byte search (strchr)	0.142	0.243	0.198	0.028
Substring (strstr)	0.057	1.273	1.030	0.088
Thread creation/joining	0.248	0.126	45.761	0.142
Mutex lock/unlock	0.042	0.055	0.785	0.046
UTF-8 decode buffered	0.073	0.140	0.257	0.351
UTF-8 decode byte-by-byte	0.153	0.395	0.236	0.563
Stdio putc/getc	0.270	0.808	7.791	0.497
Stdio putc/getc unlocked	0.200	0.282	0.269	0.144
Regex compile	0.058	0.041	0.014	0.039
Regex search (a{25}b)	0.188	0.188	0.967	0.137
Self-exec (static linked)	234µs	245µs	272µs	457µs
Self-exec (dynamic linked)	446µs	590µs	675µs	864µs

Figure 2.1: Comparison between musl and other C/POSIX standard library implementation for Linux [5].

Standard libraries need to be safe and secure, as systems which rely on unsafe implementations of the standard C library which aren't well built can open the system to a lot of security vulnerabilities.

2.1.3. Init system

Linux is just the kernel part of an operating system, which is the reason why we have distributions. You cannot really use Linux by itself, you need system applications to be able to boot and operate. Distributions is simply a bundle of software combined with a kernel (in this case it would be Linux for Linux distributions). Getting a Linux to boot requires an init system,

which initialises the system during the boot process. Without an init system, Linux is not able to boot itself to user-space. One of the most common init systems for Linux is `systemd`, which is not simply just an init system – but a suite of utilities which also simplifies system administration [6].

Init systems started in BSD², where a process simply launched a list of processes on startup. The processes launched by an init system are usually services, or programs such as a display manager (if the system has a graphical interface). Although init systems, as much as it sounds, doesn't handle the process of launching applications that “run on startup” after logging in. On Linux, that's generally the task of the desktop environment [7].

The `systemd` init system has been met with a lot of criticism, generally because it considers itself a suite of software rather than a simple init program. This violates one of the main concept of the Unix philosophy, which is “Make each program do one thing well.” This is similar to the concept of minimalism, but towards software [8].

There are many philosophies and arguments that some people in the Linux, and generally the Unix community, hold religiously. We'll try to be rational when talking about these issues.

Apart from the initialisation process, `systemd` also handles DNS³ resolution (`systemd-resolved`), system logging (`systemd-journald`), terminals (handling TTY, with `systemd-terminal`), power management (`systemd-pm`), boot manager (`systemd-boot`, previously `gummiboot`), network management (`systemd-networkd`), network time sync (`systemd-timesyncd`), device management (`systemd-udev`), and much more. Traditionally, all these systems and functions would not be involved with the init system [9].

The amount of tasks and system components that `systemd` seems to implement seems to gradually expand. It is important to note that not all of the tasks above is done with the main process (the main `systemd` process, which just like all init systems, has the process ID of 1), but the tasks are separated to different processes and binaries (like `systemd-networkd`), so the argument that `systemd` is one monolith process that does all these tasks (which is a commonly criticised fact) is invalid. Although, the argument that no such suite of processes, which are heavily dependent on each other, should be accepted is still a valid point.

²Berkely Software Distribution

³DNS stands for Domain Name Server, which is what resolves domain names to IP (Internet Protocol) addresses.

Having heavily independent programs on the system means that you are not able to easily swap these applications without removing all of them (although you have some flexibility with `systemd`, there are some components which you may not swap out, or swapping them would limit the functionality of `systemd`). When using a regular init system, you are generally able to pick and install which implementation of system services you might want to use. This allows you to individually audit these applications and pick a secure implementation, something which might prove to be somewhat difficult with `systemd`.

An issue with `systemd` are the number of vulnerabilities (or specifically, CVEs⁴) attributed to it. This is somewhat expected, as `systemd` re-implements software that has existed for many years or decades in a short period of time, which has bugs and security issues ironed out.

One of the benefits of `systemd` is usability – to a certain degree. This is because the system can be configured using a more unified way of configuration and system commands can modify the system better, as the components are well integrated.

Due to the high usability of `systemd` and distributions which are based upon it, `systemd` will be used as the init system in the generated systems this project produces. Although it would be great to have this part of the system interchangeable depending on the requirements of the user, having a swappable initialisation system would be a tremendous task which this project would not implement. The implementation of such feature would possibly be a research project on its own.

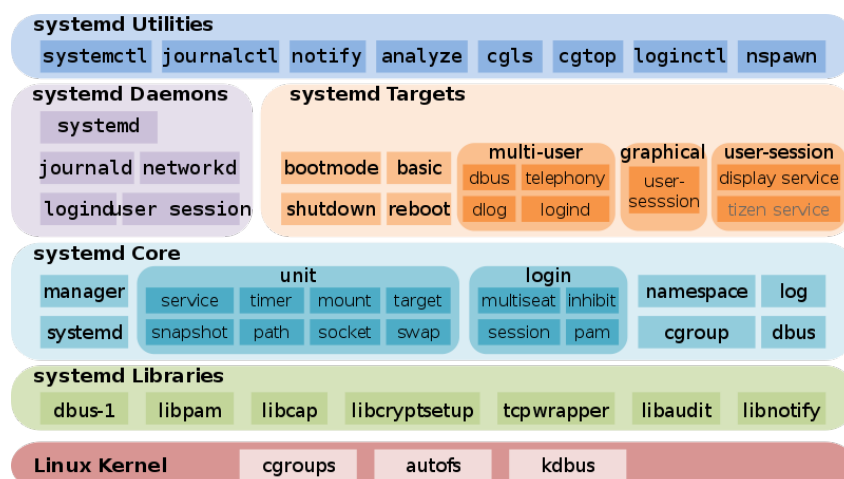


Figure 2.2: Architecture of `systemd` [10]

⁴CVE stands for Common Vulnerabilities and Exposures, which is used to refer to specific vulnerabilities and identifies them with an identifier.

2.2. Specific-Purpose Linux Distributions

There are many specific purpose Linux distributions, and more are being created. An example is RoboBuntu, which is an Ubuntu-based Linux distribution for robotics students and developers. To summarise, RoboBuntu included a custom repository which had robotics-related software, some system tweaks, and a “graphical restyling” [11]. Mancini et al. utilised tools such as the Ubuntu Customisation Kit (UCK) and Reconstructor, which are currently discontinued tools. There are similar tools, but most of them seem to be out of date and unmaintained.

Another similar project, Lin4Neuro, aims to create a Linux distribution for neuroimaging analysis. It includes 12 different neuroimaging analysis software bundled with the system, and it also aims to be lightweight and to boot quickly from a live image. Remastersys was used in creating Lin4Neuro, which is also discontinued [12].

PhyLIS is another Linux distribution, which is also a respin of Ubuntu mainly for phylogenetics and phyloinformatics⁵. Thomson stated that there are existing Linux distributions such as Bio-Linux and SciBuntu, but noted that they are a general system that lacks packages and tools for phylogenetic analysis. PhyLIS also aims to be a lightweight operating system which can run smoothly on cheap systems with multi-core processors [13].

There are major similarities between RoboBuntu, Lin4Neuro, and PhyLis. These can be summarised to the following:

- Lightweight system which can run on under-powered systems.
- The ability to use the system without installing it permanently on disk (using Live CD).
- The system bundles relevant software packages, tools, and repositories.

The aim of this project is to allow these kinds of projects to be built easily. Since all the specific purpose Linux distributions mentioned previously are no longer maintained, this project should allow the authors of those projects to create a modern version of the systems.

Penguin’s Eggs is currently one of the most recently updated remastering tools, which is a command-line based utility written in Typescript/Node. This tool allows you to redistribute your system’s installation as an installable image (as an `.iso` file). It utilises the Calamares

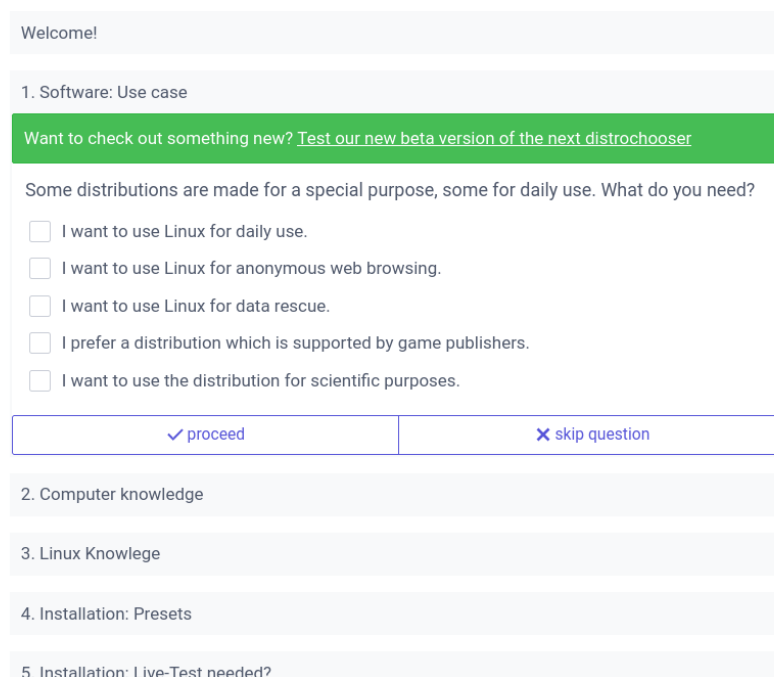
⁵Study of evolutionary relationships between organisms and the information system that stores these relationships.

installer to allow the live image to be installed on the system. Calamares will also be used in this project [14].

MX Linux, a midweight Linux distribution, has multiple remastering programs. MX has an application that allows you to update the kernel on a installation disc image, a live USB maker (a remastering tool), and RemasterCC which allows remastering the installation disc image [15]. MX Linux is the top viewed distribution on DistroWatch (as of April 2021) – a website which measures popularity of distributions (using page hit ranking) and provides news related to releases of Linux distributions.

Penguin’s Eggs and other (now defunct) remastering tools are convenient if you are planning to redistribute your system’s configuration. These tools also assume that you are familiar with the command-line and some Linux concepts, but it doesn’t help users build a custom Linux distribution if they don’t have an already-configured system to redistribute.

2.3. Linux Surveys



The screenshot shows a web-based quiz interface for 'distrochooser'. It starts with a 'Welcome!' message. The first question is '1. Software: Use case'. Below the question is a green banner with the text 'Want to check out something new? Test our new beta version of the next distrochooser'. The main question is 'Some distributions are made for a special purpose, some for daily use. What do you need?'. There are five radio button options: 'I want to use Linux for daily use.', 'I want to use Linux for anonymous web browsing.', 'I want to use Linux for data rescue.', 'I prefer a distribution which is supported by game publishers.', and 'I want to use the distribution for scientific purposes.'. At the bottom of the question area are two buttons: '✓ proceed' and '✗ skip question'. Below the question area are five more sections: '2. Computer knowledge', '3. Linux Knowledge', '4. Installation: Presets', and '5. Installation: Live-Test needed?'. The interface is clean and modern with a light gray background and blue accents.

Figure 2.3: A screenshot of distrochooser quiz.

This section would cover Linux surveys which currently exist. Unfortunately, there is only one that is available on the Internet, which is a survey which allows you to find a suitable Linux distribution.

Distrochooser is a website which allows users to find a Linux distribution which suits their need by filling a quiz or survey (shown in [Figure 2.3](#)). Each Linux distribution listed on the quiz has a weight for each of the question on the quiz, which is then used to pick the distribution that most suits their need.

The website also allows users to solve the questions in any order, skip or delete questions, and to even weight properties. Since this survey is non-linear, all the questions would be shown to all users.

A new beta version of Distrochooser was released, a significant change is that it would show you if you selected an option which would logically contradict a choice on a previous question. This would not be a problem with our design, as we are building a linear wizard which hides irrelevant questions [16].

Distrochooser is the only quiz or survey which asks questions regarding Linux distribution preferences was available online. There might be some similarity between Distrochooser's questions, and questions in our wizard shown in [Section 3.3](#). This is because both questionnaires have the similar aims.

2.4. Hardening Linux

Linux distributions have varying degrees of security, with most trying to balance security and usability for the general user. As software become more complex, so does the distributions. Hardening is a process of making a system more resistant to exploits and vulnerabilities. This can be achieved by configuration, specific security software (such as intrusion detection systems), system patches, and more [17].

There are many security-focused Linux distributions, but most of which have a specific purpose or for use on hostile environments. A popular choice is Qubes OS, which utilises virtualisation to create sandboxes. Qubes OS is considered a Converged Multi-Level Secure system, where users may interact between multiple applications freely, while they are isolated in sandboxes. It also aims to make this as usable as possible, but this system naturally comes with additional friction [18].

Theodore de Raadt, the founder and leader of the OpenBSD project – which is a security-focused operating system based on BSD, noted that “*x86 virtualisation is about basically placing*

another nearly full kernel, full of new bugs..." [19] Theo is a proponent of writing correct and robust code in the first place, instead of sandboxing vulnerable systems – which is a patch that doesn't solve the root problem.

The UK National Cyber Security Centre (NCSC) provides guidance for organisations on securing their EUDs (End User Devices). They currently provide platform-specific guidance for systems such as Android, macOS, Windows, Chrome OS, and Ubuntu [20]. The platform-specific guidance is essentially a list of recommendations that the organisations should apply to their systems and devices, which are aimed to satisfy their 12 EUD security principles [21].

NCSC currently has a secure configuration guide for Ubuntu 18.04 LTS (Long Term Support). Although it is a couple of years outdated, most of the guide is still relevant – and it may be adapted to newer versions of Ubuntu. Examples of recommendations are password strength/quality requirements, boot hardening, automatic updates, software restrictions (limiting where software can be executed), system file permissions, and more [22].

OpenSCAP – an organisation managed by Red Hat, also provides a guide on how to secure Ubuntu 18.04. They provide a long checklist of recommendations, which are categorised as either system settings, or application services settings [23].

Security vulnerabilities is also an issue that needs to be factored in. Vulnerabilities in software are inevitable, and if found in a component such as the Linux kernel, it allows attacks to gain access to the system or bypass protections [24]. Modern Linux distributions, especially those which are backed by corporations, are quick to patch vulnerabilities and have dedicated security teams. For example, the Ubuntu Security team is responsible for auditing software on their repository, tracking vulnerabilities on a global CVE database [25].

Security is a proactive practice, it is not something that is done once or after a security breach. Getting the latest security patches on the system is one of the basics of proactive security. Using a system that does not receive security patches or software updates is dangerous, which is something we take note of.

With this project, we'll aim for security through simplicity. By avoiding software and services on the system which the user might need, and including resources and tools which help users set-up their system securely. We'll also tighten the security of the system based on what is required by the user, by applying recommendations from NCSC, and other organisations.

2.5. OS Customisation & Provisioning

There are Linux tools for provisioning, management, and deployment of Linux systems. A common tool used is Ansible by Red Hat, which allows administrators to manage systems with infrastructure as code (IaC). IaC allows provisioning systems through configuration files (or other means of configuration), Ansible employs this through Playbooks [26].

These tools enable infrastructure automation, and can be combined with other tools such as version control systems and continuous integration [27].

This paper aims is not to replace these provisioning tools, but to augment them. Packer by HashiCorp is an example of a tools which augments provisioning tools such as Ansible or Terraform (HashiCorp's IaC tool). Packer is tool which automates building machine images based on a configuration file [28]. While Packer aims to create images for use in virtual environments and single-board computers, this project aims to create a web application that allows creating installation images of desktop Linux distributions.

Current modern Linux distributions utilise package managers and configuration tools which follow an imperative model, so changes to the system is stateful. This makes irreversible changes to the system, without the ability to roll back.

NixOS is a Linux distribution with a package manager (Nix) which builds an entire system based on “a purely functional specification.” Other Linux distributions utilise package managers and configuration tools which follow an imperative model, so changes to the system are irreversible. NixOS on the other hand, due to its stateful nature, allows the system to trivially roll back to previous configured state. The system is atomic during upgrades and system changes, so the system would never be in an inconsistent or broken state due to an upgrade. NixOS design comes with other benefits, for example, due to the nature of the package manager, the system avoids dependency hell⁶. There are many other benefits which is out of the scope of this research [29].

We'll focus on the Nix expression language, which is used in creating the system specification. This makes it possible to build the static⁷ parts of the system from a purely functional

⁶Dependency hell is when different programs depend on different versions of a system library, causing a conflict. This is also referred to “DLL Hell” on Windows.

⁷Static parts of the system includes things such as “software packages, configuration files and system startup scripts” [29].

specification language [29].

```
1 # Configure the boot loader.
2 boot.loader = {
3     systemd-boot.enable = true;
4     efi.canTouchEfiVariables = true;
5 };
6
7 # Setup networking and firewall.
8 networking = {
9     hostname = "serow";
10    wireless.enable = true;
11    firewall = {
12        enable = true;
13        allowedTCPPorts = [ 8080 ];
14        allowedUDPPorts = [ ];
15        allowPing = false;
16    };
17    interfaces = {
18        enp0s31f6.useDHCP = true;
19        wlp0s20f3.useDHCP = true;
20    };
21 };
22
23 # Enable sound.
24 sound.enable = true;
25 hardware = {
26     u2f.enable = true; # yubikey
27     pulseaudio.enable = true;
28     bluetooth.enable = true;
29     cpu.intel.updateMicrocode = true;
30 };
31
32 # Configure i18n and console font
33 i18n.defaultLocale = "en_US.UTF-8";
34 console = {
35     font = "Lat2-Terminus16";
36     keyMap = "us";
37 };
38
39 # Set the timezone.
40 time.timeZone = "Asia/Dubai";
41
42 # System packages to install.
43 environment.systemPackages = with pkgs; [
44     # system packages
45     bc
46     wget
47     neovim
48     bat
49     #...
```

```
50 ];  
51  
52 # Configure zsh  
53 programs.zsh = {  
54   enable = true;  
55   enableAutosuggestions = true;  
56   enableCompletion = true;  
57 }
```

Code 2.1: A system configuration defined in the Nix expression language

As shown in [Code 2.1](#), we can configure everything from the boot loader, networking, what system packages to install, and how they should be configured. This expression language doesn't have a front-end – such as a graphical user interface (GUI) which allows users to install packages or configure software. Users must be familiar with writing Nix expressions and how a system may be described. Although this system is powerful, the prerequisites makes it unusable for users who don't have experience with functional programming and Linux system internals.

NixOS is not going to be used as a basis for this project, but we can learn a lot from how Linux systems can be described with the Nix expression language, as we too, are building a Linux system from a machine-readable specification.

2.6. OpenBSD & Proactive Security

OpenBSD prides itself as a security-focused operating system, based on BSD (Berkeley Software Distribution). They emphasize on “portability, standardization, correctness, proactive security and integrated cryptography.” [\[30\]](#)

OpenBSD has a security auditing team which audit code and resolve any bugs or security vulnerability they find. They have an intense auditing discipline, where code may be audited multiple times, by different members of the team. And they are proactive when it comes to resolving issues, even if a discovered issue is not yet known to be exploitable [\[31\]](#).

They also have security features not found on the other BSDs or Linux, or adopted security standards early on. OpenBSD was one of the first common operating systems to enable position-independent executables (PIE) back in 2013. The developers are also not afraid to break compatibility if it brings a more security.

Pruning and polishing is a process in which OpenBSD developers identify what obsolete

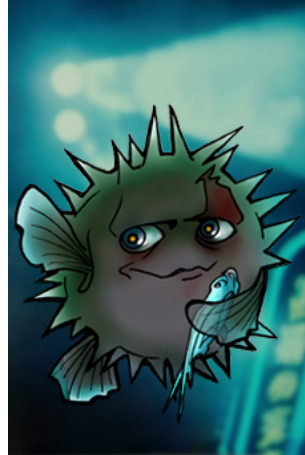


Figure 2.4: A picture of Roy Puffy (OpenBSD 5.3 Release Cover, which is the release which enables PIE by default)

code to remove (prune), and polishing existing code to increase the quality of the project.

Proactive security, and their pruning and polishing strategy is something to keep in mind when working on this project. But another question that comes to mind is why shouldn't we base this project on OpenBSD rather than Linux? Although OpenBSD is one of the most secure operating systems, it isn't very usable for people who aren't familiar with Unix-based or Unix-like systems.

This brings us to usable security. Since this project is aimed for usable security to beginners and non-technical users, OpenBSD cannot be used as a base.

2.7. Usable Security

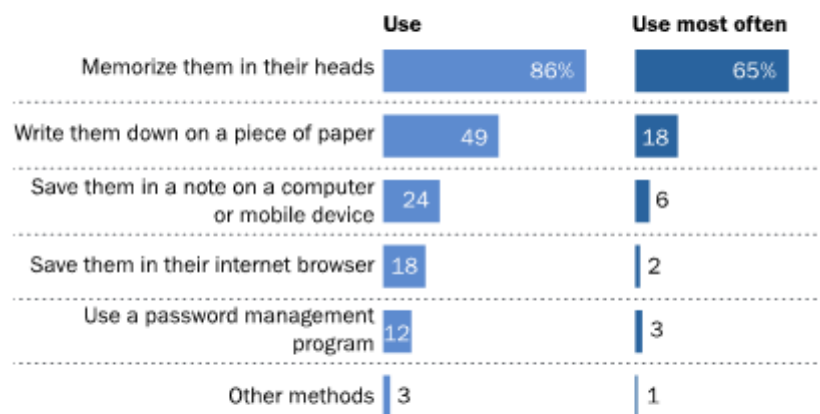
The field of usable security has been studied for over two decades, as security-related interfaces are usually not initially built with usability in mind. Many security tools start with an aim of security, without focusing on making the tool usable for the general user – which is usually the case with command-line tools or obscure programs which often look like prototypes. Developers should move towards making these tools more accessible to the general public.

Having higher security requirements can reduce usability, and might encourage counterproductive workarounds [32]. Security usually comes with inconvenience, as General Benjamin Childlaw (a US Air Force officer) said “If you want security, you must be prepared for inconvenience.” It usually seen as a burden, which adds friction when doing work. Security is a continuous and iterative process, and usability should be factored in [33].

Let's take password managers as an example. There are many password managers users can choose from, yet the adoption rates are low. Only 12% of Americans use a password manager, while the majority memorise the passwords, or write them down on a piece of paper (see Figure 2.5) [34].

Most Americans keep track of their online passwords by either memorizing them or writing them down

% internet users who keep track of their online passwords in the following ways



Note: Results for "use most often" category include those who use only one technique to manage their passwords.

Source: Survey conducted March 30-May 3 2016.

"Americans and Cybersecurity"

PEW RESEARCH CENTER

Figure 2.5: Survey conducted by Aaron Smith and Kenneth Olmstead on techniques used to manage passwords.

Many of the users that do not use a password manager are not aware of better solutions. Some users are also confused about the browser save password prompts, not sure whether they are part of the website, browser, or their computer. Even if some of these users are aware, some of the users believe that it is not worth the hassle of setting up a password manager. There are also other points that are raised, such as security and single point of failure concerns. Removing barriers from tools that improve security is important, security experts might know about these tools, but an average person might not. Combined with a low perception of a security risk, this leaves adoption rates low [35].

Password managers is one example, but the same concept can be applied to different topics, such as two-factor security. Reducing barriers is a large part of usable security, including informing and educating the user. Educating users is also a challenge, how would you design a

system that would educate users on good security practices? Knijnenburg and Cherry conducted a study on using comics for security (and privacy) notices. They concluded that using comics made topics that would generally be boring and technical, into a “*fun and engaging activity*.” [36]

The wizard will aim to allow users to make informed decisions regarding security policies, and to maximise their benefits without encouraging counterproductive workarounds. Future work can be done in further informing the user regarding security practices, either with documentation, training, comics, or other forms of material.

2.8. Conclusion

There are interesting concepts we can learn from. The aim of this project is not to generate the most secure operating system, but it is to allow users to make informed decisions and create a system that matches their use case and security requirements. We also discover different techniques, such as OpenBSD’s pruning and polishing process, and Linux hardening techniques that we can implement in our project in some ways.

Chapter 3 Prototype Development & User Interface

This section aims to describe the prototype and initial design of the system that was developed in this project, including details on why these decisions are made.

3.1. Overview

The web application being developed is an easy to use system which allows anyone, without in-depth Linux knowledge, to create their custom Linux distribution that fits their use case. This can be used by students, organisations, or companies – although it isn't aimed to any target group. It is supposed to be a tool, or a building block for everyone regardless of their technical knowledge.

The user will initially be greeted with is a dynamic wizard, which is essentially a survey that would show different questions depending on the selections to previous questions. The aim of this wizard is to collect user requirements to build the Linux distribution.

Once the specification is collected from the user, the system may make design and security decisions (security stringency). This is then be used in generating the Linux distribution. This is described in a machine-readable specification (we selected JSON), which is passed to the builder application.

Furthermore, the system may allow advanced users to apply their own custom changes to the systems, such as adding scripts to the build process – which allows endless customisability. Although this is an optional feature for advanced users.

We'll employ progressive enhancement when building the web application, this would allow us to make a system that is fast, accessible, and reliable. We'll also follow Web Content Accessibility Guidelines (WCAG)¹ to support the widest range of users.

¹WCAG is a range of web development guidelines, which helps users with disabilities, such screen reader users.

3.2. Requirement Analysis

This chapter covers the requirement analysis of the system. In both functional and non-functional requirements, each requirement contains an identifier, a priority (see [Section 3.2.1](#)), and a description of the requirement. The description is split into two parts, a summary followed with a longer explanation (shown in *italics*). A column showing completion status has been added, showing whether it has been implemented in the final project implementation.

Objectives are also cited for each requirement, those objectives may can be referred to back in [Section 1.2](#).

3.2.1. Priority Scheme

Each requirement will follow the MoSCoW method for prioritisation. A summary of this priority scheme is shown in [Table 3.1](#).

M	“Must have” – A requirement that must be satisfied upon the project completion.
S	“Should have” – A critical requirement.
C	“Could have” – A desired requirement, but not necessarily required for the project’s success.
W	“Won’t have” – A considered requirement, which might not be implemented.

Table 3.1: MoSCoW Priority Scheme explained [37].

3.2.2. Functional Requirements

Each functional requirement (shown in [Table 3.3](#)) includes a functional requirement (**FR**) ID, a summary, which is followed by a longer explanation of the requirement shown in italics.

3.2.3. Non-functional Requirements

Just like the functional requirements, the non-functional requirements (shown in [Table 3.4](#)) also includes a non-functional requirement (**NFR**) ID, a summary, and a longer detailed explanation.

3.2.4. Requirements and Objectives

This section will cover the relationship between the requirements and the objectives (found in [Section 1.2](#)). This relation is shown in [Table 3.5](#).

3.3. The Wizard

The wizard contains a set of questions that a user may be asked. This is shown in [Section 3.3.1](#). This is just the initial set of questions, further questions may be added during the implementation phase as further research is conducted. A screenshot of how a question is displayed on the prototype is shown in [Figure 3.1](#).

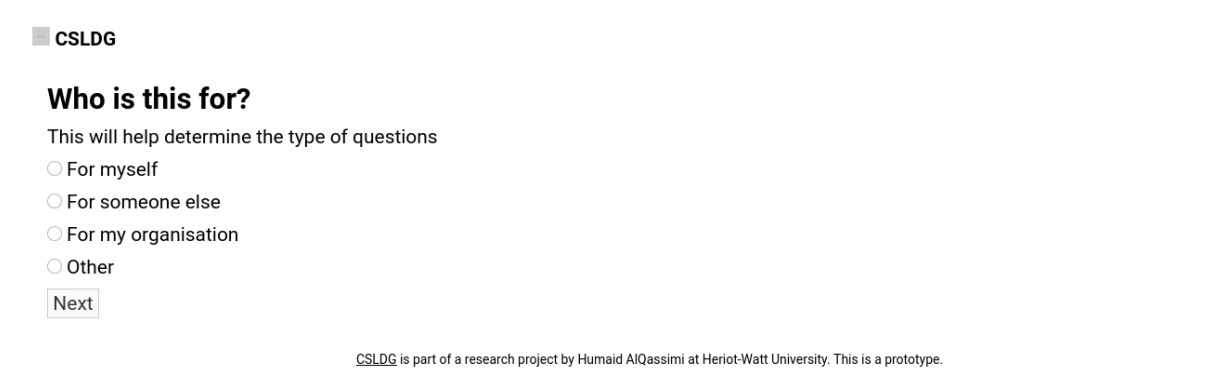


Figure 3.1: Screenshot of the prototype wizard.

3.3.1. Questions, Options & Effects

QID 1: Who is this for?

Options:

- For myself
- For someone else
- For my organisation
- Other

This would be used to change the wording of the wizard. May be later used for further things, such as further branding for organisations.

QID 2: What is your use case?

Options:

- Student/School
- Office

- Studio (Audio, Graphics, Video)
- Computer Science/Development
- Other

This would affect the applications which would be bundled with the distribution. For example, if the use case is development, development tools (such as compilers) would be bundled with the system.

Below is a list of the package bundles for each option.

- **Student/School:** gcompris, kdeedu, geogebra, scratch, tuxmath, gnome-dictionary, inkscape, mypaint, gimp, xournal, gbrainy, tuxtype, eoptes, eoptes-client, calibre, vym, freeplane, gnome-sound-recorder, audacity, rocs, atomix, anki
- **Office:** gnucash, ofxstatement, ofxstatement-plugins, gimp, inkscape, gnome-dictionary, xournal, audacity, thunderbird, timeshift
- **Studio (Audio, Graphics, Video):** obs-studio, audacity, kdenlive, inkscape, mypaint, handbrake, lmms, blender, youtube-dl
- **Computer Science/Development:** clang, make, cmake, bat, audacity, gimp, inkscape, mypaint, ffmpeg, bvi, curl, dust, feh, ffmpeg, build-essential, licenser, jq, pandoc, plantuml, shellcheck, youtube-dl
- **Other:** *Let the user pick later.*

QID 3: How would you rate your Linux knowledge?

Options:

- Beginner: I need help with some/most tasks
- Moderate: I know how to manage most things myself
- Expert: I usually know how to fix things myself

This would change the complexity of the questions asked. If the user is a beginner, more assumptions would be made. If the user is an expert, they will be shown more advanced options.

QID 4: What is your security risk factor?

Options:

- Low: I am not a targeted individual
- Medium: I am in a medium risk environment
- High: I am currently being targeted

Depending on the security risk, more security measures would be enabled. This is factored with **QID 5**.

QID 5: How much do you care about usability?

Options:

- High: I really care about usability, and I don't mind if it affects usability
- Medium: I don't mind if some security measures affect usability
- Low: I don't care if security drastically affects usability

For usable security, the user must be willing to accept usability compromises. Otherwise they might make counterproductive workarounds.

This is factored with answer from **QID 4**.

QID 6: Do you prefer to use the terminal/shell?

Options:

- Yes, I prefer to use the terminal
- I don't mind using the terminal
- No, I'd always to use a graphical interface instead

This would be hidden if the user selected *Beginner* for **QID 3**. The system might be bundled with more utility tools if they prefer the terminal.

If **Yes** is answered, the system will bundle the following list of programs: `tmux`, `imagemagick`, `curl`, `units`, `inxi`, `jq`, `mlocate`, `pv`, `ranger`, `sxiv`, `feh`, `screen`, `nmap`, `lm-sensors`, `ffmpeg`, `zathura`, `zathura-pdf-poppler`

QID 7: Will you be running random binaries or scripts from the Internet?

Options:

- Yes, I'd like to be able to run scripts or binaries from the Internet
- No, I don't want to be able to run scripts or binaries from the Internet
- I'm not sure.

This will decide whether the home directory partition would have the `noexec` flag, disallowing any code from executing in the home directories (such as the Desktop or Download folder).

This is one of the NCSC recommendations.

3.3.2. Security Stringency Score

Questions 4 and 5 shown in [Section 3.3.1](#) determines the security settings of the generating distribution. Each answer is given a score count. For question 4, low risk has a score of 0, while high risk has a score of 2. And for question 5, high usability has a score of 0, while low usability has a score of 2.

The final score is a scale out of 4, we will refer to this as the *stringency score*. Where a score of 4 has the most security restrictions.

Depending on the score, the system will impose stricter configuration and security measures. An example of the changes is shown in [Table 3.6](#). If the stringency score is higher than 0, then it'll include the security measures in the scores less than it. For example, if the stringency score is 2, it also includes measures listed for scores 0 and 1.

The list of measures listed in [Table 3.6](#) is not comprehensive, and would ideally continue to expand.

ID	Pri.	Description	Completed?
FR 1	M	The system shall have a web interface. <i>This web interface would contain the wizard and configuration options for generating the Linux distribution.</i>	Yes
FR 2	S	The system shall be able to track users without having to log-in, allowing them to save their progress. <i>Users should be able to immediately start the wizard without having to create an account initially.</i>	Yes
FR 3	M	The system shall contain a dynamic wizard, which collects the Linux distribution user specification. <i>The dynamic wizard would show/hide specific questions based on previous answers. E.g. if the user selected an expert level, they get detailed questions.</i>	Yes
FR 4	S	The system should allow the user to further customise the system after the wizard. <i>After the dynamic wizard, the user should be shown further customisation options, such as for security-related configurations</i>	Yes
FR 5	M	The system shall be able to generate a custom Linux distribution, providing an installable .iso image file. <i>The system should have a 'builder' component which generates/builds a Linux distribution based on a given specification.</i>	Yes
FR 6	C	The user shall be able to choose or upload a custom Linux distribution name and logo. <i>This should be an option after the wizard is completed.</i>	Mostly
FR 7	S	The user shall be able to see what kind of changes will be applied to the system. <i>Before the build process – which may take a while, the user should be able to see a summary of what kind of settings, or changes the system will have.</i>	Yes
FR 8	C	The user should be able to add custom files and scripts in the system build process. <i>Custom files may be graphics or documents which the user wants to be pre-bundled with the system. Adding the option for scripts in the build process allows further customisations by advanced users.</i>	Yes
FR 9	S	The system should be verifiable by the user, to see what changes have been applied to the system from the base system (i.e. Ubuntu). <i>Since the sensitivity of operating systems is high, having a way to verify that the system generated is untampered with is essential.</i>	Yes
FR 10	M	The system generated should be secure (according to policies generated by the wizard, etc). <i>The wizard would collect user preferences and security settings, and the system generated should be based according to those selections. These may be part of the NCSC EUD guidelines, or other security guidelines and practices.</i>	Yes
FR 11	W	The system should allow users to share configurations publicly. <i>There could be some marketplace where users can share their system configurations.</i>	No

Table 3.3: Functional Requirements.

ID	Pri.	Description	Completed?
NFR 1	M	The web application shall have a clean, uncluttered, and minimal interface. <i>Keeping the web application minimal helps with accessibility and users on slower or limited Internet connections. It further helps usability by having an uncluttered, and straight-forward interface.</i>	Yes
NFR 2	M	The system shall accommodate for users without knowledge security or Linux internals. <i>We cannot assume that all users have knowledge on security and Linux systems. We have to accomodate these users by showing only relevant questions during the wizard, and hide overly technical aspects for these users.</i>	Yes
NFR 3	W	The system be optimised to speed up the build process using caching and other techniques. <i>The build process may be cached, or be built in a process that allows reusability, which will speed up the build process.</i>	No
NFR 4	M	The web application should follow Web Content Accessibility Guidelines (WCAG). <i>This will allow the system to be accessible to users with disabilities, such as low vision, colour blindness, etc.</i>	Mostly

Table 3.4: Non-functional Requirements.

Requirement	Objective Name					
	Policies	Usable Sec.	Inf. Pol.	Wizard	Builder	Integrity
FR 1	X		X	X		
FR 2				X		
FR 3	X	X	X	X		
FR 4				X		
FR 5					X	
FR 6						
FR 7						X
FR 8						
FR 9						X
FR 10	X	X				
FR 11						
NFR 1						
NFR 2		X				
NFR 3						
NFR 4						

Table 3.5: Relationship between the objectives and requirements.

Stringency Score	Measures
0	<ul style="list-style-type: none"> Remove analytics packages (<code>whoopsie</code>, <code>apport</code>, <code>popcon</code>)
1	<ul style="list-style-type: none"> Firewall (with <code>ufw</code>) Password Quality Check (moderate, with <code>pam_cracklib</code>)
2	<ul style="list-style-type: none"> Automatic TTY timeout Make home directories not world-readable (<code>DIR_MODE=0700</code>, <code>UMASK=077</code>) Firefox tweaks (adblocker with uBlock Origin)
3	<ul style="list-style-type: none"> Disable shell access for new users by default Check boot partition integrity on boot (<code>hashboot</code>) Reduce screen lock time Firefox tweaks (require HTTPS for all websites, litterboxing, DNS over HTTPS, only load websites using strong cipher suites) Require tougher password quality Require full-disk encryption Allow lockdown of USB inputs (<code>usbguard</code>)

Table 3.6: Security measurements depending on the security stringency score.

Chapter 4 Web Implementation

4.1. Overview

This chapter will cover implementation – what tools are used, why were they used, and the process of implementing the features of the web user interface.

This chapter will not cover the builder system implementation, as it would be separated into its own dedicated chapter (see [Chapter 5](#)).

4.2. Choosing Appropriate Tools

There are many parts to the project. But we can simply split the implementation of this project into two parts; the web interface and the builder process.

For both implementation we have primarily used the Go programming language, mostly due to familiarity, good web frameworks, and safety.

The web interface is written with plain HTML and CSS without any modern web frameworks around it. The web pages are generating using Go's templating engine. This simplicity helps us increase the usability to more conform to WCAG (Web Content Accessibility Guidelines) recommendations, and allows the web pages to load fast.

4.3. Development Environment

Since we are creating and/or modifying a Linux distribution, the system has to run on a Linux system. This is because in order to customise our Linux distribution, we have to execute processes inside the customised Linux environment – which would require complex emulation if we were not to run the program on a Linux system. This emulation could cause significant performance inefficiencies.

The type of Linux system doesn't matter as much. I'm running the builder system (as defined in [Section 5.4](#)) on my system, which is running Void Linux – a Linux distribution which

doesn't use `systemd` as the init system, or `apt` as the package manager [38]. And yet, the system is running fine. This is because the process we need to run (`apt`) can run on systems which don't require `systemd`. Running more complicated processes might require a fully fledged Debian-based (or at least Ubuntu) system running with `systemd`, although we have not yet faced with this limitation.

There was an issue (described at the end of [Section 5.4.2](#)) regarding domain name resolution. This was due to `systemd-resolved` had a symbolic link set up to a file under the `/run` directory, which is a temporary filesystem that is mounted on boot on a system with `systemd`. We had to work around this issue, by removing the symbolic link and replacing it with a file. We also removed `statoverride`¹ file stored by `dpkg` (`apt`'s backend), which was causing an issue. Removing the file made `dpkg` independent on some system services which was causing `apt` to exit with an error.

4.4. Architecture

The system is split into two parts, the web interface and the builder. The web interface is implemented as a web server, which can be exposed on a public servers where users are able to visit and interact with the wizard and customise a Linux distribution. This is shown in [Figure 4.1](#).

Both parts of the system are independent processes which communicate with each other. They both share the same system storage which allows the generated file to be served by the web server process.

The system architecture is shown in [Figure 4.1](#), and the diagram shown in [Figure 4.2](#) shows how a user would interact with the system.

4.4.1. Interaction between web and builder components

The builder component is available as a web API, which is only accessible to the web interface. The files are available on a shared disk/directory, which the web component is able to serve from.

¹Full file path is: `/var/lib/dpkg/statoverride`

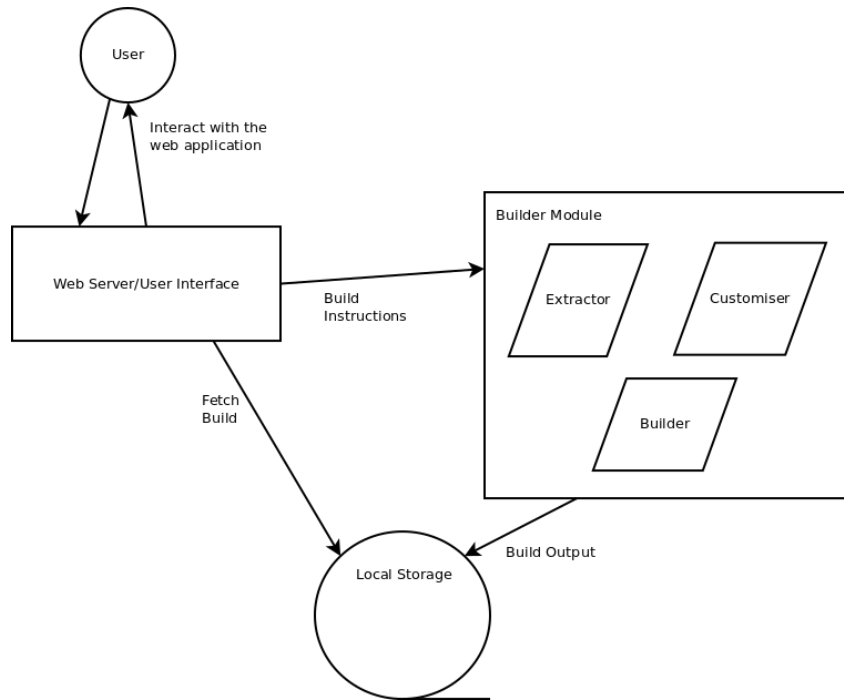


Figure 4.1: Architecture Diagram

4.5. Web Interface

4.5.1. Packages Listing

Getting the list of packages that users may install or uninstall was a challenge. Ubuntu has over sixty-two thousand packages in its repositories, which we have to show to the user. Showing all sixty-two thousand packages on a single page would be intimidating, as most of the packages are system libraries and dependencies.

We had to build a procedure which gets the list of all packages that are available on Lubuntu, whether they are pre-installed and their categories.

Getting a list of all packages that are available, and which are pre-installed is simple. We can run about two commands², one which gets a list of all available packages, and the second gets the list of installed packages. Both lists are overlaid to create a list of packages with a flag of which are already installed.

Next, we go through each package and fetch the category name. This is the most time-consuming step, as it takes a second to get the categories of about 40-50 packages. The entire

²The commands are run in the `chroot` environment of the distribution. Commands are `apt list`, and `apt list -installed`.

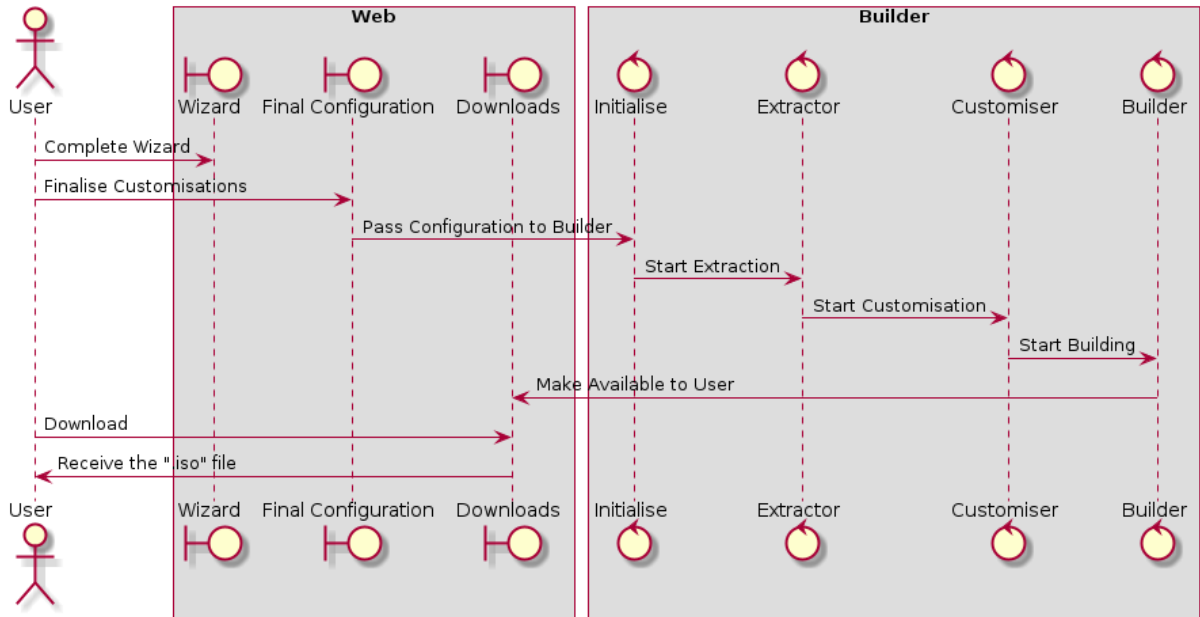


Figure 4.2: Use Case Diagram of the System

process took about half an hour to complete. A potentially faster way is to parse the entire `apt` repositories, but that would require us to write our own parser for reading `apt` package specifications.

Once we fetched all the information we needed, we store that in a manifest file which we can retrieve later.

The section names used for categorisations are not user-friendly, as sections have names like `universe/games`, `multiverse/math`, and so on. Some of them are of the same category but split into multiple sections. To make this user-friendly, we created our own categorisations which are bundles of sections. For example, the “Development” category will include packages from `universe/utils`, `universe/tex`, `utils`, `multiverse/javascript`, `java`, and so on. An icon is also given to each category to make it easier to identify (seen on [Figure 4.4](#)).

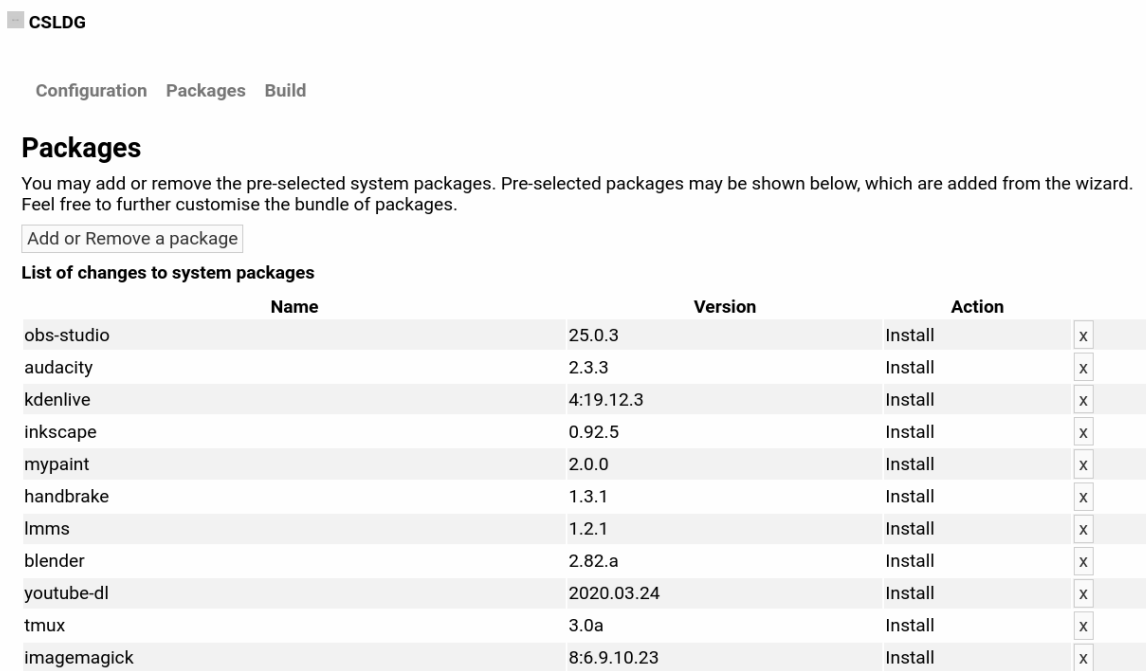


Figure 4.3: Packages selection on the web interface.

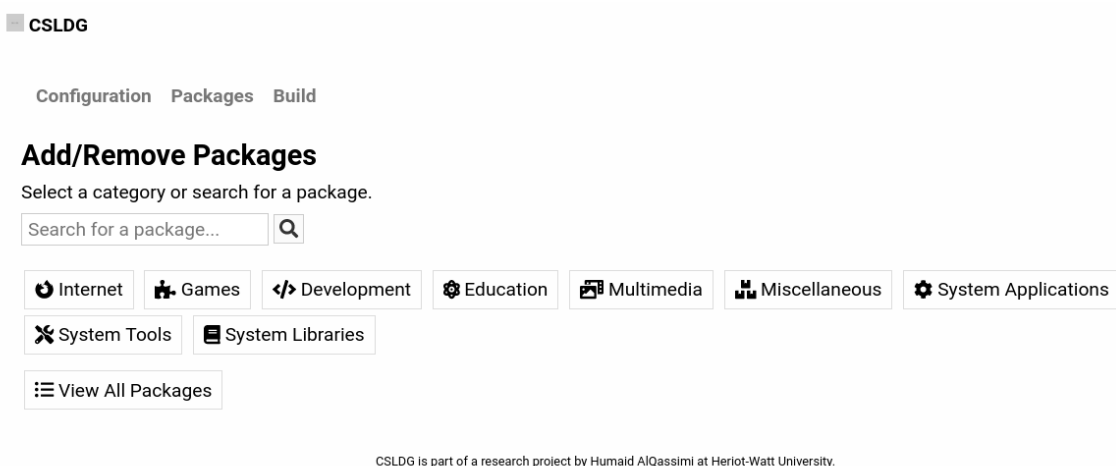


Figure 4.4: Packages categorisation on the web interface.

Chapter 5 Builder System Implementation

5.1. Overview

The aim of the builder system is to create the custom distribution based on the user preferences in the wizard and configuration pages. This chapter will cover the builder system – the module which creates the distribution for the user. We'll go into technical details and challenges with the development of this module.

5.2. What is a Distribution

Linux operating systems come as a multitude of “distributions”, this is because Linux is just a kernel (the low-level system that handles processes and hardware). Linux doesn't really come with user-space applications, which is what you interact with on a daily basis. This is the reason why you can't really install plain “Linux.”

This is where distributions come in, a distribution is a system image which contain the user-space applications, an init system¹, and all the core utilities and applications. Every distribution comes with a different bundle and configuration of software, and usually have different purposes. There are many general purpose desktop distributions (like Ubuntu or Fedora), and some of them have “server” versions of them (like Ubuntu Server) which is a stripped down version of the desktop edition which is more catered for server use. But you could find distributions that are more catered to specific uses other than desktop and servers, like what is previously listed in [Section 2.2](#).

The system image you usually get when downloading a Linux distribution is an ISO image file. The “ISO” in the name simply refers to the ISO standard (ISO 9660) of the file format. It simply contains all the data an optical disc can contain in a single file, which is useful if you are burning the image to a disc (to create an installation disc). Although this is a disc format, the contents can also be burned to a USB memory stick, which is more common these days.

¹An init system is the program that loads the important system processes on boot, and handles the startup and shutdown sequence of the system.

The ISO image file contains a boot loader, and file and flags that allows the disc to be bootable. This allows you to boot into the system that exists in the disc. Most importantly, the image also contains the file system of the distribution, a boot loader without a system to boot into would be useless.

The Linux system that is embedded in the installation disc is special, as it contains software that can automatically install itself to the system's drive. It works by basically self-replicating the system installed on the installation disc to the system's drive, and setting up and configuring the boot loader to boot into the newly installed system.

5.3. Our Base

We are not going to build a Linux distribution from the ground-up, because that would be undertaking a huge task and that's out of the scope of our project. Our plan is to allow users to create a custom secure Linux distribution based on their preferences, this can be achieved by using a pre-existing distribution and tweaking it to our needs based on the user's preferences.

As we found in Section 2.2 of our background research, many users prefer a lightweight system especially when building specific-purpose systems. This also increases security by reducing complexity. We have chosen Lubuntu (a lightweight "flavour" of Ubuntu, a popular Linux distribution) as our base for this project. We picked Lubuntu as it has a lightweight desktop environment, and still contains the extensive repository of software (as it uses Ubuntu's repositories).

Every distribution generated by this system would be based on Lubuntu, and all customisations, configurations, and re-branding would be made on top of it.

5.3.1. Legal Considerations

As explained previously, a Linux distribution is a bundle of many applications. This also means each application may have different license and conditions. This means that Linux distributions are considered aggregate work. Lubuntu usually comes bundled with free and open-source software, but it may contain proprietary packages. It is important that these non-free² packages are removed. Otherwise, redistribution free and open-source software is allowed to the nature

²"Free" in this case refers to free and open-source software, not free as in free beer.



Figure 5.1: A screenshot of a Lubuntu system.

of their licenses.

We are basing off of Lubuntu, which is owned by Canonical Limited. Canonical Limited owns the intellectual property of Ubuntu and its official derivatives (including Lubuntu). Their intellectual property policy states that “Any redistribution of modified versions of Ubuntu must be approved, certified or provided by Canonical if you are going to associate it with the Trademarks. Otherwise, you must remove and replace the Trademarks and will need to recompile the source code to create your own binaries.” [39]

This means, the custom Linux distributions generated by the project must remove the Ubuntu trademarks. Although this policy states that binaries should be recompiled, Zimmerman (Ubuntu’s previous CTO) clarified that packages with the name ‘ubuntu’ may not need to be replaced [40].

5.4. Builder Process & Tools

Customising a Linux distribution is a daunting task even for those who are experienced with Linux and the inner workings of it. We have reviewed some tools that allows customising Linux during the background research in Section 2.5. We are going to build our own builder system from scratch, not based on any system previously made – although the process might be similar to some existing solutions that might be available. Building our own system is important, because the kind of changes we are making should be fully automated based on preferences from the web UI. This is not possible using any pre-existing tools.

Understanding the process on how this is made was really difficult, there was no single up-to-date resource to figure out this building process. The Ubuntu wiki has a guide on creating an Ubuntu-based derivative distribution, although the guide contains outdated information and hasn't been updated since 2009-2010. This was still an important resource in figuring out how to create a customised distribution [41].

A second important resource was the “Pop!_OS ISO production” repository, which contains scripts that builds *Pop!_OS* – a custom Ubuntu-based distribution created by System76 (an American Linux computers manufacturer). Their repository helped me understand the steps involved in building a customised distribution based on Ubuntu [42].

After many trial and errors, refining the processes and scripts, and prototypes, we simplified the build process into three separate parts. These can be categorised as:

1. Extract: Extract the contents of the installation image (like unzipping).
2. Customise: Apply customisations needed to the file system, which may require executing application in the file system.
3. Build: Build back the installation image based on the customised file system in the previous step.

Let's dig into each step and discover what tools are used in each.

5.4.1. Extraction Process

The extraction process is when the contents of the installation image (ISO) file is extracted from its “image” form. There are two layers of extraction.

The first layer is the installation image, which is usually an “.iso” file. This can be mounted just like a physical disc. Once mounted, the contents are copied out of the image to a directory we’ll call `extract/`. The contents of the installation image mostly is the boot loader and other things that allows the system to be booted on machines.

The actual Linux distribution’s file system is found in a single file (which in our case is located in “`casper/filesystem.squashfs`”). We’ll exclude copying that file over to the `extract/` directory, and we’ll instead directly extract that to a new directory we’ll call `chroot/`.

This process is simplified above in some ways, our implementation of the builder runs on a Linux system to simplify the process (as not to require emulation). We use the standard `mount` utility to mount the .iso installation image file, the `rsync` utility to copy over the files to the `extract` directory, and finally the `unsquashfs` utility (which comes bundled with the SquashFS Tools application) to extract the `filesystem.squashfs` file.

5.4.2. Customisation Process

Now that the file system and the installation disc files are all extracted, this makes it possible to modify the files and apply our own customisations.

Debranding

As explained previously in sub Section 5.3.1, we have a legal obligation to de-brand the Lubuntu system (to a certain degree). We first go through different files and replace the Lubuntu branding to the name of the distribution provided by the user.

Below is an inclusive list of the files in the Linux file system which contain branding that has to be modified (not including the boot loader files):

- `/etc/issue`
- `/etc/issue.net`
- `/etc/legal`
- `/etc/lsb-release`
- `/etc/os-release`
- `/etc/calamares/branding/lubuntu/branding.desc`

And in the image installation disc files:

- `/.disk/release_notes_url`
- `/.disk/info`
- `/isolinux/txt.cfg`
- `/boot/grub/grub.cfg`
- `/boot/grub/loopback.cfg`

User-specific Customisations

Once this is done, we continue to apply system customisations based on the user's preferences that was selected on the web UI.

Next, we'll install/remove packages specified by the user. To achieve this, we have to run commands where the "root" directory is set to be the extracted Linux file system. This allows us to interact with the binaries and run them as if they are running on our host machine.

Ubuntu (and generally all Debian-based distributions, which Ubuntu is based on) uses the `apt` package manager (formerly named *Aptitude*), which is a program that allows users to install, remove, and update packages among other things. This is done through a repository with mirrors hosted around the world.

Although it is recommended by Canonical that forks of Ubuntu host their own repositories, it is not required – especially as this is a research project rather than a commercial solution.

We also apply system configuration changes depending on the security stringency score (as was defined in [Section 3.3.2](#)). The list of configuration changes that will be applied is listed in [Table 3.6](#).

The user-provided script would also be executed with `bash` in the `chroot` environment.

Running apt Under a chroot Environment

To run the `apt` utility (in order to install/uninstall packages), we will have to basically fool `apt` into thinking that the extracted Linux file system is the root directory (the main `'/'` directory of the system). This is done using the `chroot` (change root) utility.

Before using the `chroot` directory, we have to mount (or *bind*) some system directories to the Linux file system directory. Generally, these would be `/proc`, `/dev`, and `/sys` directories. This allows us to run applications with `chroot` without issues, by sharing the host system resources.

We initially used `proot` (a wrapper around `chroot`) to automatically mount these directories for us, as it is simpler. Compare the two examples with [Code 5.1](#) and [Code 5.2](#)³. `proot` is a very convenient tool, but unfortunately we ran into two functionality-breaking bugs with it – one of which was resolved with a workaround⁴. Using `chroot` and mounting the directories manually resolved this issue.

```
1 proot -R /path/to/chroot
2   -w /
3   -b /proc/
4   -b /dev/
5   -b /sys/
6   -0 /bin/bash
7 # customisations script may be either piped to proot or provided
   in the filesystem to be executed
```

Code 5.1: Example of customisation with `proot`

```
1 mount --bind /dev/ /path/to/chroot/dev
2 chroot /path/to/chroot /bin/bash
3 # and in the chroot environment:
4 mount -t proc none /proc
5 mount -t sysfs none /sys
6 mount -t devpts none /dev/pts
7 # customisations can be done from this point on
```

Code 5.2: Example of customisation with `chroot`

Furthermore, we faced an issue regarding network access in this environment. Network (or more specifically, Internet) access is required for us to install new packages on our system. This is required by `apt`, as without access to its repositories, it cannot install new packages⁵.

Specifically, this issue was with domain name resolution, which is generally handled by `systemd-resolved`. Since in our `chroot` environment we aren't running any services, we will have to work around getting the programs we need to work without any service running.

³Please note that the functionality of both examples are not 100% identical, especially in regards to what is being mounted. Although this variance does not seem to be the source of the issue we were facing.

⁴These are issues [#106](#) (workaround available) and [#182](#) on `proot`'s repository on GitHub.

⁵We are referring to packages which are not available as a cache or locally on the system, which may be installed with `apt` or `dpkg` without network access.

With older versions of Ubuntu, we can simply overwrite the `/etc/resolv.conf` file – adding a “`nameserver`” field with an address that is a valid DNS server. This allows applications in the `chroot` environment able to correctly resolve domain names (such as `archive.ubuntu.com`, the host name of the server which contains the software repositories for `apt`).

Creating and Running or Customisation Script under `chroot`

Once we are able to create an environment we can run `apt` with, we can work on generating some kind of script that allows us to automate installing and removing packages, or performing other changes to the system we need.

To do this, we generate a script and write it to the file on the extracted Linux file system. Once it is generated, we can use the `chroot` command to run `bash` (as a shell script interpreter) to run our script we generated.

The customisation script starts with setting some environment variables, defining the root directory and locale. We then mount `none` to `/proc`, `/sys`, and `/dev/pts`. This sets up a pseudo-filesystems that allows us to run processes without issues in the `chroot` environment. These pseudo-filesystems exist on a normal system, and are created/mounted automatically. But since this is not a system we are running, we have to manually mount them.

We next update `apt`'s local repository cache, which is important as the software in the local repository cache might be out of date – causing errors when attempting to install software. The script so far is shown in [Code 5.3](#).

```
1 export HOME=/root
2 export LC_ALL=C
3
4 mount -t proc none /proc
5 mount -t sysfs none /sys
6 mount -t devpts none /dev/pts
7
8 apt update
```

Code 5.3: Snippet from the beginning of the customisation script

We then add commands to install/remove the packages as given to the builder module. This is in the format of: `apt install -y <package name>`, where `<package name>` is the name of the package to be installed. This is the same with uninstalling packages, which is done in the format of: `apt purge -y <package name>`.

Once this step is done, we run: `apt autoremove -purge -y`. This removes all of the “orphan” packages, which are the packages which were installed as dependencies of previous packages, which we no longer need. This helps reducing the size of the final generated file.

The `-y` argument is added to all the `apt` commands, which allows us to skip the confirmation prompt (shown usually as `[Y/n]`) when performing system changes.

Finally, we unmount all the pseudo-file systems we mounted in the beginning, which is shown in [Code 5.4](#).

```
1 umount /proc
2 umount /sys
3 umount /dev/pts
```

Code 5.4: Snippet from the end of the customisation script

This generated script file is placed in `/root/` (the `root` user’s home directory) temporarily in the extracted Linux filesystem. We then use `chroot` to run this script using `bash` (the shell script interpreter).

5.4.3. Building Process

Now that all the changes are made to the extracted image and file system files, we can start to prepare to build back the bootable ISO image.

Since some packages may have been removed or new packages have been installed, we have to regenerate the manifest file (“`filesystem.manifest`”), this file simply contains the list of packages that are installed on the system. We have to then make a copy of it with the name (“`filesystem.manifest-desktop`”), which is the same as the previous file with just the live CD-specific software removed from the list – as this is the list of packages that are supposed to be in the final install of the system. In our case that would be the `calamares` (the system installer wizard program) and `casper` (related to the live system setup).

There are many of these changes that you have to make so the image is in a correct “state”.

The manifest file is created in the Linux file system, and this is all that’s needed to be changed in the file system during the building process. So we can “squash” back the file system using the `mksquashfs` utility.

During development, we found that this `mksquashfs` step took longer than any other step. Changing the compression method to `lz4` drastically reduced the time this step took.

Once this step is complete, we have to create a file containing a size of the filesystem `squashfs` file, and `md5` checksum of the files on the boot image. These are used in the boot process to check the integrity of the image.

The final step is to create the ISO image file, for which we use the `xorriso` utility. Once the ISO image file is create it, the file is hashed and signed, and these files are provided to the user.

Chapter 6 Evaluation

6.1. Evaluation Strategy

This chapter will cover system evaluation strategy. We define two ways to evaluate this system, the first part ([Section 6.1.1](#)) will cover technical evaluation of the system. The second part ([Section 6.1.2](#)) will cover the usability study.

6.1.1. Technical Evaluation

Technical evaluation is mostly a programmatic approach to evaluating the system. This involves mostly testing the system, whether it could produce a correct image file, and also whether the wizard system works without issues.

This makes the technical evaluation split into two parts, the wizard evaluation ([Section 6.1.1](#)) and the image evaluation ([Section 6.1.1](#)).

Builder Evaluation

We initially intended to use unit testing to test this part of the system. But during the development and testing process we found this to be infeasible, as most parts of the builder module require super-user (administrator) privileges on the system. The builder module could easily destroy the host (the system the application is running on), which happened during development, where the process was recursively deleting mounted partitions in the `chroot` directory. This ended up removing some important system files from my computers, which was luckily resolved after reboot¹.

This could have easily went worse, and it is not something that I would risk with unit testing. Especially as unit tests are a series of independent “units” being tested, many testing frameworks can run tests out of order – which is something that could break the process. And it would be useless to have just one monolithic unit test to test the entire build module.

¹Luckily, the `/dev` directory tends to re-generate on reboot.

So instead of unit testing the builder module, we created a *test* command which runs a specific set of build instructions. This command has been used intensively during the implementation process, and help us test each feature and iron out the issues.

In the future, it may be nice to have some sorts of unit testing which does not rely on super-user privileges – possibly something that runs in a sandbox. Unfortunately, Docker is not able to run the builder tool, as it doesn't support the `chroot` utility. Although solutions with QEMU is still possible, albeit a bit heavier and slower.

We did not expect this type of complexity with unit testing this module, implementing a safe unit testing function for the builder module would require us to discover or invent a new way to sandboxing unit tests.

Image Evaluation

The validity of the system image produce at the end of the process should also be evaluated. Since this couldn't be done using unit tests mentioned in [Section 6.1.1](#), a separate testing suite should be created for this purpose.

The image files could be run with QEMU, which is a system emulator to run the image files generated in a virtual environment. This allows us to boot into our systems and test its functions. QEMU is useful for manual testing, but it can't be used for automated testing without building an entire automated test suite on top of it – which is currently infeasible for this project. The QEMU Machine Protocol would be a start if we would ever decide to create an automated test suite with QEMU.

There are other solutions to check the validity of the system, which is by checking the integrity of the image. This can be done by inspecting the disk image file's contents in a programmatically. By mounting the different partitions of the system (there should be two, the boot and the system partition), we can proceed to verify that only the intended files were changed, and everything else was untouched. Mounting the partitions would also act as a test on whether the disk image file is valid. This would allow us to evaluate the **Builder** objective.

This validity test could be taken even further, allowing the user to independently verify the system (see **FR 9** in [Table 3.3](#), allowing us to meet the **Integrity** objective).

Unfortunately, due to the complexity of generating and booting the custom Linux distri-

bution, we are not able to evaluate the usability of the final product. Hence, we are not able to evaluate **Usable Security** objective.

6.1.2. Usability Evaluation

The usability evaluation would cover the usability, which is aimed at evaluating the user interface and experience of the system. The system which is being evaluated is the wizard (shown in [Section 3.3](#)). The aim of the system to be usable even for users who are not familiar with the process of creating a Linux distribution, or even with the technicalities of Linux. The wizard should also help user make informed decisions regarding security policies (meeting the **Policies** and **Informed Policies** objective).

The full survey is posted in [Appendix A](#), and a summary of the anonymised data is posted in [Appendix B](#).

The study is an online questionnaire, which also collects information regarding their experience with Linux and computers in general. This would help gauge the difficulty of the system for users with varying levels of technical expertise.

The usability evaluation does not contain testing the Linux system, because getting the participants to install virtualisation software is a huge barrier. Also this would require the system to be fully implemented. This might have been possible with physical setup, where participants can use a provided computer to perform the study on. Although this was not possible due to the COVID-19 pandemic.

6.2. Evaluation Results

6.2.1. Usability Results

We had 29 participants in our usability study, and 20% of the participants were female. Overwhelming majority of the participants were in the 18-24 age group (except one in 25-34 age group).

The average SUS score is 64, which is considered below average [43]. Getting the average scores of the users who are familiar with securing Linux or Unix systems (answering 4 or higher on the Likert scale) reveals a slightly lower score (61), which we can assume as in the margin

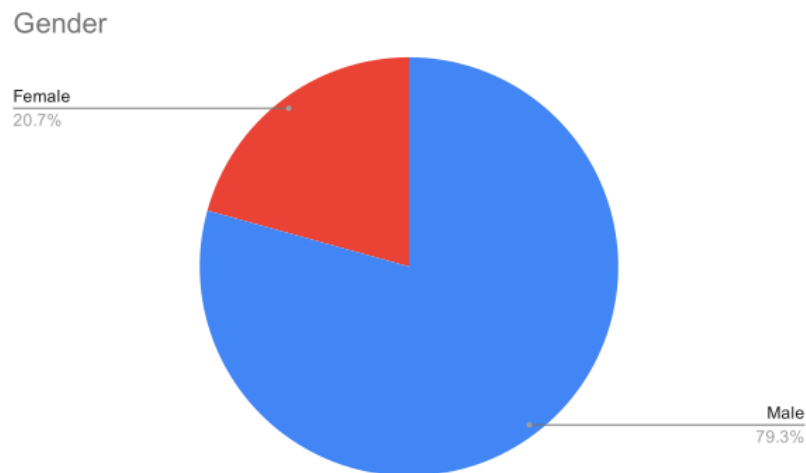


Figure 6.1: Gender of participants

of error.

Participants' think that they are likely to use this system, and their opinion is slightly more higher (4.1 vs 4.38 out of 5) after trying out the system.

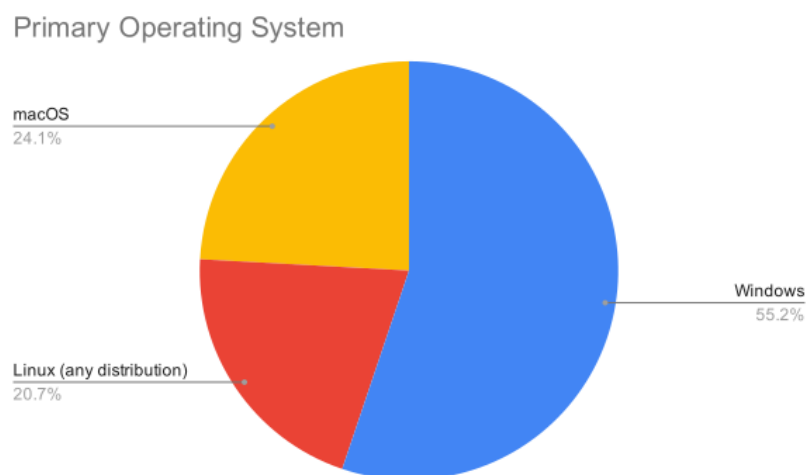


Figure 6.2: Primary Operating System of participants

Participants generally appreciate the customisability, naming their custom distribution, and the ease of use.

Although participants wanted more advanced questions for more advanced users with very specific requirements. Some users wanted more information on what each question in the wizard does, as they find the simple description not detailed enough.

The participants were provided a prototype, the later developed version contains more

options on what is being customised, and allows the users to see and pick what packages get installed on the system.

This was an insightful study, although the SUS score was below average. Compared to other tools that are available (which were listed in [Section 2.5](#)), this is probably way higher in terms of usability, this in and of itself is quite significant. The tools that already exist requires expertise in the field (excluding the now defunct OpenSUSE).

There are still ways to improve the accessibility of this system, the goal of this project is not to have the most usable web UI – although usability is quite important, which we have achieved.

There is low to no correlation between a person’s confidence in managing Linux system and their consideration of using the system (see [Figure 6.3](#)), which is a positive result. If we consider the low negative correlation, it is still a positive result, but we might include more options for power users (which we did after the study, such as scripting options).

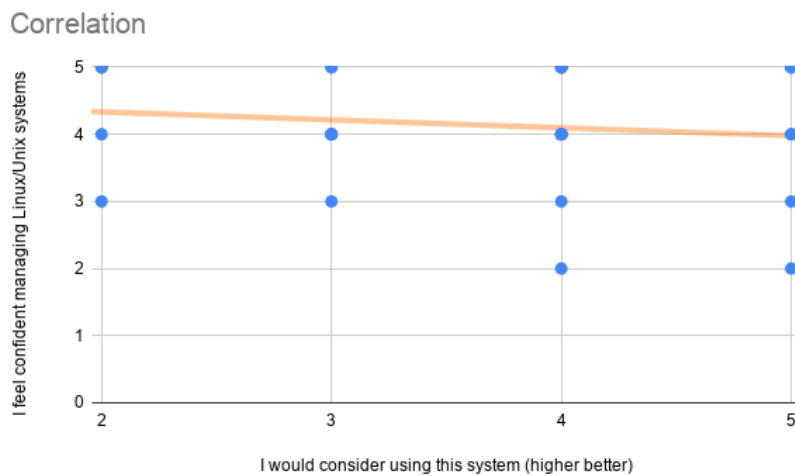


Figure 6.3: Correlation with Linux/Unix confidence and consideration of using the system

Future plans is to make the system more usable, and the product it generates more secure. Although we improved the system after the study (we implemented the prototype into a working product), improving the usability and security of the entire system (the web UI and the generated system) is an iterative process.

Chapter 7 Conclusion

7.1. Overview

This project implemented the customisation process of Linux distribution, and the different ways we can improve upon it. With security and usability being important objectives of this project, we looked at combining different solutions. The focus is not only to improve the usability (more specifically, in terms of usable security) of the generated Linux system, but also we look into improving the process of creating a Linux distribution.

We have seen the current solutions that currently exist, and how they don't provide the main objectives that we have stated. This is a unique project, which would change the way special-purpose Linux distributions are built. We have discovered previous research papers where a solution like this would be useful.

A system which builds a Linux distribution without requiring user interaction during the build process is a significant achievement. There are no current tool that is able to build a custom Linux distribution dynamically. Not only this system is able to create a custom distribution without user input during the build process, but it also doesn't require technical background and has a focus on usability in both the wizard and the generated system.

7.2. Limitations & Future Work

The scope of this type of project is vast, there are many ways to dig into and improve.

7.2.1. More Customisability

Although the system allows you to customise the Linux distribution to some extent using the web interface, more work can be done in expanding what kinds of things that could be customised.

Currently, in addition to user-friendly options, there is a shell script field that allows the user to apply their own changes to the system. Although this is a great option to keep for

advanced users, we should reduce the need for users to manually write a script by adding more user-friendly options to the web interface.

7.2.2. Better Usability & User Experience

The system's current usability is adequate based on the usability study, and we have improved upon the system drastically after the feedback. But there are still parts where the usability and user experience can be enhanced.

For example, selecting which software packages to install is an important part of the system. Currently, it shows the packages in separate categories to make it easier to browse. Furthermore, the system could provide a simpler interface – such as the ones seen on modern “app stores.”

No product is perfect, and there are always opportunities to improve on the usability and user experience of any system.

7.2.3. Further Improvements in Usable Security

Security is a forever evolving issue, especially when it comes to usability. More improvements can be made with improving usable security, such as by adding more security improvements based on further questionnaires (questions in the wizard).

This project implemented a security stringency score system, which may model the usability and security field a bit too trivially. Further work can be done into expanding how the usable security space is defined.

7.2.4. Human Factor

Many breaches happen due to human factors, which is an important issue in the field of security. Even if the system is secure, a person who operates systems that isn't well trained in cybersecurity basics can be a huge risk.

More can be done regarding educating users to reduce the risk of the human factor.

7.2.5. Security Interface

In the future, it would be a good idea to have a security dashboard. The security dashboard should allow you to do tasks such as enable or disable security features, view audit logs, setup

a USB lockdown mode (and whitelist specific USB devices), view failed login attempts, see the boot partition integrity, notify against CPU vulnerabilities (such as meltdown or spectre), and more.

There doesn't currently seem to exist a security dashboard like this, and having it would increase system usability and security for those who don't know how to secure and manage Linux systems manually.

References

- [1] *The Linux Kernel Archives*. URL: <https://www.kernel.org/> (visited on 04/05/2021).
- [2] *Chapter 1. What is Linux?* Linux Sea. Oct. 3, 2018. URL: https://web.archive.org/web/20181003201630/http://swift.siphos.be/linux_sea/whatislinux.html (visited on 04/05/2021).
- [3] The Austin Common Standards Revision Group. *POSIX 1003.1 Standard FAQ*. URL: https://www.opengroup.org/austin/papers/posix_faq.html (visited on 04/05/2021).
- [4] Rich Felker, et al. *About musl*. musl. URL: <https://musl.libc.org/about.html> (visited on 04/05/2021).
- [5] Eta Labs. *Comparison of C/POSIX standard library implementations for Linux*. URL: https://www.etalabs.net/compare_libcs.html (visited on 04/20/2021).
- [6] systemd Contributors Lennart Poettering. *systemd Homepage*. systemd. URL: <https://systemd.io/> (visited on 04/05/2021).
- [7] Yvan Royon and Stéphane Frénot. “A Survey of Unix Init Schemes”. In: *arXiv:0706.2748 [cs]* (June 20, 2007). arXiv: [0706.2748](https://arxiv.org/abs/0706.2748). URL: <http://arxiv.org/abs/0706.2748> (visited on 04/05/2021).
- [8] Eric S. Raymond. *The art of UNIX programming*. Nachdr. Addison-Wesley professional computing series. OCLC: 552065285. Boston: Addison-Wesley, 2008. 525 pp. ISBN: 978-0-13-142901-7. URL: <http://www.catb.org/~esr/writings/taoup/html/ch01s06.html>.
- [9] John E. Vincent. *The End of Linux*. blog dot luis. Sept. 23, 2014. URL: <https://blog.luis.org/blog/2014/09/23/end-of-linux/> (visited on 04/05/2021).
- [10] Shmuel Csaba Otto Traian. *Systemd Architecture*. Sept. 29, 2013. URL: https://commons.wikimedia.org/wiki/File:Systemd_components.svg (visited on 04/11/2021).
- [11] A. Mancini et al. “RoboBuntu: A Linux distribution for mobile robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2009, pp. 2544–2549. DOI: [10.1109/ROBOT.2009.5152548](https://doi.org/10.1109/ROBOT.2009.5152548).
- [12] Kiyotaka Nemoto et al. “Lin4Neuro: a customized Linux distribution ready for neuroimaging analysis”. In: *BMC Medical Imaging* 11.1 (Jan. 25, 2011), p. 3. ISSN: 1471-2342. DOI:

- [10.1186/1471-2342-11-3](https://doi.org/10.1186/1471-2342-11-3). URL: <https://doi.org/10.1186/1471-2342-11-3> (visited on 09/28/2020).
- [13] Robert C. Thomson. “PhyLIS: A Simple GNU/Linux Distribution for Phylogenetics and Phyloinformatics”. In: *Evolutionary Bioinformatics* 5 (Jan. 1, 2009). Publisher: SAGE Publications Ltd STM, EBO.S3169. ISSN: 1176-9343. DOI: [10.4137/EB0.S3169](https://doi.org/10.4137/EB0.S3169). URL: <https://doi.org/10.4137/EB0.S3169> (visited on 09/28/2020).
- [14] Piero Proietti. *pieroproietti/penguins-eggs: On the road of Remastersys, Refracta, System-back and father Knoppix!* URL: <https://github.com/pieroproietti/penguins-eggs> (visited on 11/16/2020).
- [15] MXLinux Contributors. *MX-19.2 Users Manual*. Feb. 8, 2020. URL: https://mxmanuals.s3.us-east-2.amazonaws.com/user_manual_mx19/mxum.pdf (visited on 05/04/2021).
- [16] Christoph Müller. *Distrochooser*. URL: <https://distrochooser.de> (visited on 11/16/2020).
- [17] trimstray. *The Practical Linux Hardening Guide*. original-date: 2018-10-06T21:57:36Z. Oct. 22, 2020. URL: <https://github.com/trimstray/the-practical-linux-hardening-guide> (visited on 10/22/2020).
- [18] Abdullah Issa, Toby Murray, and Gidon Ernst. “In search of perfect users: towards understanding the usability of converged multi-level secure user interfaces”. In: *Proceedings of the 30th Australian Conference on Computer-Human Interaction*. OzCHI ’18. New York, NY, USA: Association for Computing Machinery, Dec. 4, 2018, pp. 572–576. ISBN: 978-1-4503-6188-0. DOI: [10.1145/3292147.3292231](https://doi.org/10.1145/3292147.3292231). URL: <http://doi.org/10.1145/3292147.3292231> (visited on 11/16/2020).
- [19] Theodore de Raadt. *Re: About Xen: maybe a reiterative question but ..* E-mail. July 24, 2007. URL: <https://marc.info/?l=openbsd-misc&m=119318909016582> (visited on 11/16/2020).
- [20] UK National Cyber Security Centre. *End user device (EUD) security guidance*. NCSC.GOV.UK. URL: <https://www.ncsc.gov.uk/collection/end-user-device-security> (visited on 04/08/2021).
- [21] UK National Cyber Security Centre. *EUD Security principles*. NCSC.GOV.UK. URL: <https://www.ncsc.gov.uk/collection/end-user-device-security/eud-overview/eud-security-principles> (visited on 04/08/2021).
- [22] UK National Cyber Security Centre. *End user device (EUD) security guidance for Ubuntu 18.04 LTS*. NCSC.GOV.UK. URL: <https://www.ncsc.gov.uk/collection/end-user->

- [device-security/platform-specific-guidance/ubuntu-18-04-lts](#) (visited on 09/28/2020).
- [23] OpenSCAP, Red Hat. *Guide to the Secure Configuration of Ubuntu 18.04*. OpenSCAP Security Guide. URL: <https://static.open-scap.org/ssg-guides/ssg-ubuntu1804-guide-default.html> (visited on 04/08/2021).
- [24] Haogang Chen et al. “Linux kernel vulnerabilities: state-of-the-art defenses and open problems”. In: *Proceedings of the Second Asia-Pacific Workshop on Systems*. APSys ’11. New York, NY, USA: Association for Computing Machinery, July 11, 2011, pp. 1–5. ISBN: 978-1-4503-1179-3. DOI: [10.1145/2103799.2103805](https://doi.org/10.1145/2103799.2103805). URL: <http://doi.org/10.1145/2103799.2103805> (visited on 04/16/2021).
- [25] *Security Team*. Ubuntu Wiki. URL: <https://wiki.ubuntu.com/SecurityTeam> (visited on 04/16/2021).
- [26] Red Hat. *How Ansible Works*. URL: <https://www.ansible.com/overview/how-ansible-works> (visited on 11/18/2020).
- [27] Kief Morris. *Infrastructure as code: managing servers in the cloud*. OCLC: 951624089. 2016. ISBN: 978-1-4919-2439-6 978-1-4919-2438-9.
- [28] HashiCorp. *Introduction*. Packer by HashiCorp. URL: <https://www.packer.io/intro> (visited on 11/18/2020).
- [29] Eelco Dolstra, Andres Löh, and Nicolas Pierron. “NixOS: A purely functional Linux distribution”. In: *Journal of Functional Programming* 20.5 (2010). Publisher: Cambridge University Press, pp. 577–615. DOI: [10.1017/S0956796810000195](https://doi.org/10.1017/S0956796810000195).
- [30] The OpenBSD Developers. *OpenBSD*. OpenBSD. URL: <https://www.openbsd.org/index.html> (visited on 12/01/2020).
- [31] The OpenBSD Developers. *OpenBSD: Security*. OpenBSD. URL: <https://www.openbsd.org/security.html> (visited on 12/01/2020).
- [32] Bryan D. Payne and W. Keith Edwards. “A Brief Introduction to Usable Security”. In: *IEEE Internet Computing* 12.3 (May 2008). Conference Name: IEEE Internet Computing, pp. 13–21. ISSN: 1941-0131. DOI: [10.1109/MIC.2008.50](https://doi.org/10.1109/MIC.2008.50).
- [33] Butler Lampson. “Privacy and Security Usable Security”. In: *Communications of the ACM* (Nov. 1, 2009). Publisher: ACM PUB27 New York, NY, USA. URL: [http://dl-acm-org/doi/abs/10.1145/1592761.1592773](http://dl.acm-org/doi/abs/10.1145/1592761.1592773) (visited on 09/28/2020).

- [34] Aaron Smith and Kenneth Olmstead. “Americans, password management and mobile security”. In: *Pew Research Center*: (Jan. 26, 2017). URL: <https://www.pewresearch.org/internet/2017/01/26/2-password-management-and-mobile-security/> (visited on 04/16/2021).
- [35] Shikun Aerin Zhang et al. “Why people (don’t) use password managers effectively”. In: Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019). 2019. URL: <https://www.usenix.org/conference/soups2019/presentation/pearman> (visited on 04/16/2021).
- [36] Bart Knijnenburg and David Cherry. “Comics as a Medium for Privacy Notices”. In: Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016). 2016. URL: <https://www.usenix.org/conference/soups2016/workshop-program/wfpn/presentation/knijnenburg> (visited on 04/16/2021).
- [37] *A Guide to Business Analysis Body of Knowledge (BABOK Guide)*. OCLC: 656599335. Toronto, ON, Canada: International Institute of Business Analysis, 2009. ISBN: 978-0-9811292-2-8 978-0-9811292-1-1.
- [38] Juan RP and Void Linux contributors. *Enter the void*. The Void Linux distribution. URL: <https://voidlinux.org/> (visited on 04/11/2021).
- [39] Canonical. *Intellectual property rights policy*. Ubuntu Terms and policies. URL: <https://ubuntu.com/legal/intellectual-property-policy> (visited on 12/10/2020).
- [40] Matt Zimmerman. *Use of Ubuntu trademarks by derivatives (Re: Concerns)*. E-mail. Jan. 8, 2007. URL: <https://lists.ubuntu.com/archives/ubuntu-devel/2007-January/023107.html> (visited on 04/21/2021).
- [41] *LiveCDCustomization*. ubuntu documentation - Community Help Wiki. URL: <https://help.ubuntu.com/community/LiveCDCustomization> (visited on 04/08/2021).
- [42] *GitHub Repository pop-os/iso*. original-date: 2017-05-09T17:00:24Z. Mar. 27, 2021. URL: <https://github.com/pop-os/iso> (visited on 04/08/2021).
- [43] U.S. General Services Administration. *System Usability Scale (SUS) | Usability.gov*. usability.gov. URL: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html> (visited on 04/02/2021).

Appendix A Usability Survey

Embedded below is the survey exported from Google Forms.

The confirmation message after submission: “Questionnaire has been submitted. If you want to withdraw from the study, you have 7 days to submit your withdrawal request. You can do this by emailing Humaid AlQassimi <ha82@hw.ac.uk>.”

Secure Linux Distribution Generation Study

*Required

Consent to act as a subject in an experimental study

Principal Investigator: Humaid AlQassmi <ha82@hw.ac.uk>
Institution: Heriot-Watt University

The purpose of the study is to evaluate the user interface and experience of the system, which is a wizard that creates a Linux distribution.

The survey would be evaluated, not the Linux distribution itself. This is an online questionnaire which would also collect information regarding their experience in computer-related fields and Linux, as this is relevant to the study.

Personal data collected: Age, Gender, Computer Experience

You will not be required to install any software, and the study will be conducted online-only. Only a web browser and an internet connection would be required. Collected information would be stored securely and safely according to GDPR guidelines. If you are a student, your participation will not affect your courses or affect your relationship with the university in any way.

You are free to decline to participate in this study. You may also end the study at any time. You are able to withdraw from the study within 7 days of your participation. Withdrawal means your data will be completely removed and destroyed.

Since this is an online study, if you decide to withdraw in the middle of your participation, you may simply close browser window or tab of the study. You only need to send a withdrawal request if you have submitted the survey.

Withdrawal requests or queries may be sent to: Humaid AlQassimi <ha82@hw.ac.uk>

You must be at least 18 years of age to participate in this study. This is an evaluation study, the system is being evaluated and not you. Therefore, there is no right or wrong answers.

Please keep note of: today's date and initials used. You'll need this in case you decide to submit a withdrawal request.

1. Enter your initials *

e.g. H. A. Please do not enter your full name.

2. I certify that I have read and understood the consent form above. *

If there are any questions regarding this consent form, feel free to contact the investigator at the email listed above before you continue.

Tick all that apply.

Consent

Demographic Questionnaire

3. What is your gender? *

Mark only one oval.

- Female
 Male
 Prefer not to say

4. What is your age? *

Mark only one oval.

- 18 to 24
 25 to 34
 35 to 44
 45 to 54
 55 to 60
 Prefer not to say

5. What is your primary operating system? *

Mark only one oval.

- Linux (any distribution)
- Windows
- macOS
- ChromeOS
- BSD (OpenBSD, FreeBSD, etc)
- Other

6. I feel confident in managing Linux or other Unix systems *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

7. I feel confident in securing Linux or other Unix systems *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

About the system

The system you will be evaluating is a wizard that allows you to build a custom operating system (Linux distribution) based on your preferences. The wizard will go through the use cases and security related questions, to build a system. The operating system would allow you to create a secure operating system for yourself, your organisation, or even your family.

You are able to customise what software is installed with the system, pick a name (and even logo) for your custom operating system.

8. I would consider using this system. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

9. I would consider recommending this system to others. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

Prototype Evaluation

System Prototype

Please visit <https://csldg.humaidq.ae> and complete the following tasks:

10. Complete the wizard *

Go through the wizard, answering all of the questions.

Tick all that apply.

Completed

11. Select a custom name for the operating system *

Pick any name, doesn't have to be special. Such as "Home OS".

Tick all that apply.

Completed

12. Confirm the system specification *

Review the system specification and the changes that would be applied to the system.

Tick all that apply.

Completed

13. Build and download the system *

Find the build option to create your system, and find the option to download it. Note: Since this is a prototype, you are not downloading anything. When you get to the "download is complete" page, consider this task complete.

Tick all that apply.

Completed

Usability Survey**14. I think that I would like to use this system frequently ***

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

15. I found the wizard unnecessarily complex *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

16. I thought the system was easy to use *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. I think that I would need the support of a technical person to be able to use this system *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. I found the various functions in this system were well integrated *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. I thought there was too much inconsistency in this system *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. I would imagine that most people would learn to use this system very quickly *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. I found the system very cumbersome to use *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

22. I felt very confident using the system *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

23. I needed to learn a lot of things before I could get going with this system *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

Post Questionnaire

24. I would consider using this system. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

25. I would consider recommending this system to others. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

26. What was the best parts of the system? If there was any.

27. What was the worst parts of the system? If there was any.

28. What do you think can be improved?

This content is neither created nor endorsed by Google.

Google Forms

Appendix B Usability Survey Results

Embedded below is the survey results. The results have been simplified (SUS score calculated), and the open-ended text inputs have been removed.

Gender	Age	Primary Operating System	I am confident in managing Linux or other Unix systems	I am confident in securing Linux or other Unix systems	I would consider using this system.	I would consider recommending this system to others.	SUS Score	I would consider using this system.	I would consider recommending this system to others.
Male	18 to 24	Windows	2	3	5	5	55	5	5
Male	18 to 24	Linux (any distribution)	3	2	5	5	72.5	5	5
Male	18 to 24	Linux (any distribution)	3	3	4	4	60	4	4
Female	18 to 24	Windows	3	2	5	5	70	5	5
Male	18 to 24	macOS	5	5	5	5	80	5	5
Male	18 to 24	Windows	3	3	3	3	70	4	3
Male	18 to 24	Windows	5	4	5	5	80	5	5
Male	18 to 24	Windows	3	2	4	4	75	4	4
Male	18 to 24	Linux (any distribution)	4	3	4	5	70	5	5
Male	18 to 24	Windows	2	2	5	5	55	4	4
Female	18 to 24	Windows	4	4	5	4	72.5	4	4
Male	18 to 24	macOS	5	4	3	3	40	3	4
Male	18 to 24	macOS	3	3	4	4	50	3	4
Female	25 to 34	Windows	4	4	4	4	35	4	4
Male	18 to 24	macOS	4	3	5	5	80	5	5
Male	18 to 24	macOS	4	3	3	3	55	3	4
Female	18 to 24	macOS	2	3	4	4	35	5	5
Male	18 to 24	Windows	5	4	5	5	75	5	5
Male	18 to 24	Windows	3	2	4	2	65	4	2
Male	18 to 24	Windows	4	2	3	3	75	4	4
Female	18 to 24	Windows	2	2	3	3	60	3	3
Male	18 to 24	Linux (any distribution)	4	2	3	4	75	4	4
Male	18 to 24	Linux (any distribution)	4	3	4	5	75	5	5
Male	18 to 24	Windows	5	3	5	5	72.5	4	5
Male	18 to 24	Windows	5	5	5	5	47.5	4	5
Male	18 to 24	Linux (any distribution)	5	5	3	5	72.5	2	5
Male	18 to 24	Windows	4	2	5	5	77.5	5	5
Male	18 to 24	macOS	2	2	4	3	60	5	5
Female	18 to 24	Windows	4	4	4	4	47.5	2	4