Securing Software Development using Developer Access Control

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

at Rhodes University

Grant Ongers

Grahamstown, South Africa
March 9, 2020

Abstract This research is aimed at software development companies and highlights the unique information security concerns in the context of a non-malicious software developer's work environment; and furthermore explores an application driven solution which focuses specifically on providing developer environments with access control for source code repositories. In order to achieve that, five goals were defined as discussed in section 1.3.

The application designed to provide the developer environment with access control to source code repositories was modelled on lessons taken from the principles of Network Access Control (NAC), Data Loss Prevention (DLP), and Google's BeyondCorp (GBC) for zero-trust end-user computing. The intention of this research is to provide software developers with maximum access to source code without compromising *Confidentiality*, as per the *Confidentiality*, *Integrity* and *Availability* (CIA) triad.

Employing data gleaned from examining the characteristics of DLP, NAC, and BeyondCorp — proof-of-concept code was developed to regulate access to the developer's environment and source code. The system required sufficient flexibility to support the diversity of software development environments. In order to achieve this, a modular design was selected.

The system comprised a client side agent and a plug-in-ready server component. The client side agent mounts and dismounts encrypted volumes containing source code. Furthermore, it provides the server with information of the client that is demanded by plug-ins. The server side service provided encryption keys to facilitate the mounting of the volumes and, through plug-ins, asked questions of the client agent to determine whether access should be granted.

The solution was then tested with integration and system testing. There were plans to have it used by development teams who were then to be surveyed as to their view on the proof of concept but this proved impossible. The conclusion provides a basis by which organisations that develop software can better balance the two corners of the CIA triad most often in conflict: *Confidentiality* in terms of their source code against the *Availability* of the same to developers.

Acknowledgements

I would like to thank my wife and editor Cindi Ongers, without whose assistance this work would be complete gibberish. My utmost appreciation to Clive Crous for his very large role in the implementation of the code and for being a sounding board throughout.

Additionally, a great deal of thanks is due to Dr Yusuf Motara, my supervisor on this project, for his guidance and patience along the way and to Dr Barry Irwin for giving me the opportunity to be a part of the Rhodes information security program and for helping me beat this work into submission.

To the many voices of encouragement and to my fellow students who proposed ideas and made suggestions along the way, my thanks goes out to you too — you all know who you are.

This research is dedicated to my late mother Lesley Ongers, who always believed in me.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The core concept in Information Security, according to Death (2017) and Hernandez (2012) is the *Confidentiality*, *Integrity* and *Availability* (CIA) triad (see Figure 1). The triad describes the intrinsic conflict between the three poles of Information Security and helps explain the balance required between them in order to provide a suitable level of security for an organisation. The representation selected here succinctly illustrates the generally skewed ratio by which developers are afforded *Availability* (access) to work on source code at the cost of *Confidentiality* and possibly *Integrity* (or at the very least reduced controls) of same.

While many papers including Lee, Woo, Lee, and Joo (2019) and Duclervil and Liou (2019) explore the process of developing secure software, i.e., the process needed to ensure that the software produced is secure in nature, there are very few that envision securing the environment in which it is created, that is: the relevant local repositories of the developer in a way that is feasible for productivity. This work is focused on researching and developing a software application that will go a step further by extending secure software development to that of the developer's environment.

Developers, as a class of IT professionals, are unique in two ways. In the first instance, research identifies developers as having a complex skill set. Developers must be able to analyse, be creative (Nagy, 2019), communicate well, work as a team, be detailed orientated (as well as seeing the bigger picture), and have excellent abilities in maths, abstraction and logic (Siegmund, Kästner, Apel, Parnin, Bethmann, Leich, Saake, and Brechmann, 2014).

Secondly, the developers role within the IT environment lends itself to having direct access to crucial company information assets, and more worrisome, they are also mostly exempt

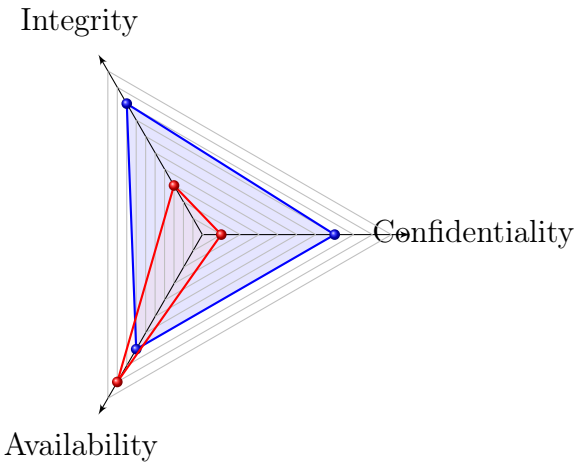


Figure 1.1: The CIA Triad as Coverage Area

from any of the controls that would normally be put in place to protect those assets. The primary reason for this is because those controls are seen as too restrictive according to Verdon (2006), stifling the creative freedom developers need to work effectively.

Software developers want the freedom to work when and where the mood strikes according to Meyer, Barr, Bird, and Zimmermann (2019) and as a result they tend to hold large quantities of source code in an unencrypted form on portable machines the moment that machine is turned on — even if the source code is not being worked on. Furthermore, they tend to work unusual hours (Meyer *et al.*, 2019), meaning that what would be considered an unusual access pattern for any other employee may not be considered as such for them. According to Salim (2017), "Another essential requirement, almost indispensable, is flex-time, and freedom to work from home. Creativity and inspiration never follow a 8 to 5 schedule." The realisation that coders often require flexible working times is not new. In the book Peopleware DeMarco and Lister (1999, p. 43) asked seminar participants to talk about their working hours. The results showed that to be productive many of them said that "... people may come in early or stay late or even try escape entirely, by staying home for a day to get a critical piece of work done." The author asserts:

> "Staying late or arriving early or staying home to work in peace is a damning indictment of the office environment." (DeMarco and Lister, 1999)

Facilitating these individuals with their special skill set and demands in order to deliver work effectively and efficiently makes implementing necessary security measures a real

challenge. On the one hand, a company does not want to stifle developers' ability to deliver code, but on the other hand, foregoing all security measures is dangerous terrain, considering the current information technology climate where security breaches are on the rise.

1.1 Scope and Limitations

The goal of this research was to review solutions that allow for the protection of source code while balancing this against the need for developers to have access to it. The scope of this research covered building a proof of concept (PoC) based off the review done and the testing of this PoC — both as a fit-for-purpose, functional solution — and, as a fit-for-use solution with development teams testing it.

The limitations of the research however, were that the success of the latter question, covered by User Acceptance Testing, rested with an external organisation. That external organisation needed to consent to — and implement — the Code Corral solution. Due to time constraints and other limitations with the intended organisation they unfortunately withdrew from the PoC. The work done to ensure the quality of the planned testing has been included in this work but the results of this testing will remain future work on this topic.

The threat model examined for this research is one without a malicious developer. In this scenario individual developers, in good faith, will comply with the security policy, insofar as they are aware of it, and any restrictions that they are reminded of, insofar as it does not overly inconvenience them. Essentially, the new security policy will not hinder their ability to work on projects they were assigned to prior to the new policy.

1.2 Problem Statement

Considering the triad of ***C****onfidentiality*, ***I****ntegrity* and ***A****vailability* (CIA): in environments where **Integrity** is managed well by version control systems is it possible to afford developers the **Availability** to source code in such a way as to allow them the required freedom to work as they do best while still affording organisations a measure of control and a feeling of security around the **Confidentiality** of their intellectual property?

This thesis explores possibilities for restoring the balance in the CIA triad, requiring relatively minor losses in granting developers *carte blanche* over their environments while securing serious gains in terms of keeping the intellectual property (IP) to which they have access protected with *Integrity* intact. It explores the question of whether it is possible to make developer environments meet a company's *Confidentiality* requirements without locking those environments down to the extent that one removes the flexibility that is usually afforded to this anomalous group of IT professionals.

The primary threat actors in this instance are malicious entities who gain unauthorised access to a development machine. The risk is identified in the fact that the machine will have code checked out to it, and SSH keys that would potentially still allow repository access in cases where the security incident management process has not yet set in. A number of scenarios where this might be the case include:

- At (hostile) border locations, where the machine's owner might be required to power on the machine and where copies of the drive might be made.
- In co-working environments, where a developer's machine may be briefly unattended (and potentially unlocked).
- When outsourcing development to third parties and partners.
- For organisations that provide development work to multiple clients on-site at those clients, they may want more fine-grained control.

In these cases protections like BitLocker would already have been by-passed as the machines are powered up and possibly even logged in.

Secondary security concerns include the ability of an organisation to place restrictions (through rules) on where, when and under what circumstances code will be accessible to an otherwise authorised developer. For example, an organisation might want to create a rule preventing access to code that is subject to export restrictions when the device is outside the country's borders. Or, perhaps an organisation that provides development services to potential competitors, they may want to ensure that potentially they can control those code bases better.

Several of the solutions that will be analysed and compared embody various and divergent threat models. This research will focus on the similarities and explain the choices made for each solution as they are dealt with.

1.3 Research Goals

The five goals of this research[1] are to:

- **Goal 1**: Identify existing solutions which can be and are used to protect IP in the form of source code. (7.2.1)
- **Goal 2**: Perform a comparative study of these solutions in terms of the problem statement in balancing the aspects of the CIA triad. (7.2.2)
- **Goal 3**: Develop a solution that offers a better integration with existing version control systems without compromising the versioning control they offer, whilst leveraging a more secure way to do access control of the source code. (7.2.3)
- **Goal 4**: Develop a solution that allows for the local repository to be protected whilst ensuring that access can be revoked. (7.2.4)
- **Goal 5**: Maintain the *Availability* for the developer to the source code without impacting productivity. (7.2.5)

1.4 Research Approach

The approach which was taken in this research was to have a functioning solution to address the unbalanced nature of secure code development. The problem statement as set out in section 1.2 details that within the software development industry there has been a lack of balance in terms of the CIA triad. The focus of the research was to develop a solution which would address the *Confidentiality* of and *Availability* to source code during development.

The research was performed using a hybrid approach to data collection. Firstly the researcher identified the current systems / tools available to address this lack of balance and those were evaluated to determine whether they addressed the problem statement. The outcome of this process was used to develop a better mechanism / system that would be more able to address the lack of balance described in the problem statement.

The proposed system was then tested in terms of its functionality and acceptance. The actual system was tested using the methodology as set out in chapter 4 in section 4.1.2

[1] with their conclusion sections in brackets

and section 4.1.3. The secondary approach would be to make use of the qualitative data collection method described in section 4.1.4 to distribute the code to and survey developers within an organisation to determine from their interaction with it how well it would do to reach user acceptability.

Due to the use of a survey there was a need to follow the Rhodes ethics approval process. This was done successfully and ethics clearance was obtained.

The collection of the data within this approach would identify whether the problem statement, as set forth by the researcher, is indeed a valid one, potentially how well any one solution addresses the problem statement, and whether the solution that the researcher has proposed is functional and accepted within the realm of code development.

1.5 Document Conventions

Commands or binary names are given in `fixed width font`. Whenever the triad of Confidentiality, Integrity, and Availability (CIA) are mentioned they will be in *Capitalised italics*.

For the purposes of this work:

- *Confidentiality* is given to mean "the controls used to ensure secrecy is maintained";
- *Integrity* to mean "all means of ensuring changes are monitored and measured"; and
- *Availability* to mean "all mechanisms put in place to assure access is possible"

Code listings have been used throughout this work and are rendered in colour for easier reading.

1.6 Document Structure

This research starts out in chapter 2 (Related Work) by evaluating the real world of the developer (section 2.2), specifically referring to writing secure code (in section 2.2.1), and on the developer's primary system for interfacing with source code, and version control systems (in section 2.2.2). It then looks at how developers view general security policies

in section 2.2.3 and section 2.2.4. The potential consequences of a breach are discussed in section 2.2.5 and a further look at a number of mitigation strategies in section 2.2.6 and section 2.2.7 follows.

Several potential solutions for access control are explored. From the commonly used, tried and tested Network Access Control (NAC in section 2.3) and Data Loss Prevention (DLP in section 2.4) to the more agile and modern ones like Google BeyondCorp (in section 2.5). The next section explores access control specific to source code in several solutions surrounding Git in section 2.6.

Chapter 3 provides the details of the proposed solution that was specifically built for this thesis. The work reviewed in chapter 2 served as context and inspiration that attempts to answer the research question. Section 3.1 explains the design and looks at the particular components required to make the solution work. It reviews design choices made, along with any trade-offs and the inspiration for the design. That section is also where the actual implementation of the solution is discussed, with details on the Free and Open-Source Software (FOSS) used, together with the code which ties it all together. Section 3.2 then examines several deployment options for the solution proposed.

Chapter 4 discusses how the solution was validated in that it meets the requirements set and looks in some detail at future work in this regard.

Chapter 5 contemplates the results while chapter 6 explores the results with an in-depth discussion.

Chapter 7 wraps things up with the conclusion about how well the solutions proposed in this work fulfil the requirements of the problem statement.

Chapter 2

Related Work

This chapter discusses a review of the literature examined by the researcher, starting with the concept of the CIA Triad in section 2.1 which is referenced frequently in this work. In section 2.2 the developer's world is explored, their unique environment is examined and so it the aspiration and need to secure the development of code. This section covers topics related to the developer's daily interactions with the code and with security. An array of sub-topics like version control systems and the use of security policies in relation to code development (something which aids in the restriction of access to code by developers in section 2.2.4) will be presented. In the following sections the researcher explores potential code breaches (section 2.2.5) and the consequences which an organisation may face when such security breaches occur. The chapter goes on to explore the importance of the idea of "separation of powers" within IT security in section 2.2.6.

In section 2.3 network access control is discussed as a method of provisioning access to network resources whilst maintaining control over the devices used for that access. Section 2.4 follows, highlighting the topic of data loss prevention as an approach to securing source code for the development industry and implementations which prevent the loss of this data. The section 2.5 discusses the zero-trust network implementation of Google's BeyondCorp. The following section (section 2.6) discusses Git-related solutions such as transparent Git encryption, `git-remote-gcrypt` and `git-crypt`. The chapter concludes with a summary of the topics explored in chapter 2.

2.1 CIA Triad

The CIA triad refers to the three recognised and widely accepted cornerstones of cybersecurity: *Confidentiality*, *Integrity* and *Availability*. The first corner of the triangle is *Confidentiality*. It is synonymous with the understanding of privacy and keeping sensitive data away from unauthorised parties. The second point of the triangle is *Integrity*, which refers to the authenticity of the data. The final corner of the triad, *Availability*, attempts to balances the equation by focusing ensuring that all hardware and software is operating in a manner that guarantees productivity (Bradford, 2016). It was previously mentioned that while the proposition of the triad is that it aptly points out vital areas of security, in practice the three corners of the triad are rarely equally balanced. More often than not, the *Confidentiality* aspect is diluted for the sake of *Availability*. Companies err, traditionally, on the side of granting too much access so that they do not affect their day to day business activities, as the cost of potentially not providing enough security. That is to say that if productivity is at stake, security is often forfeited.

2.1.1 Confidentiality

> "IT service providers have an abundance of valuable information at their disposal. Intellectual property. Credit card numbers. Personal information belonging to customers. Protecting sensitive data is vital to information security and satisfying strict compliance regulations." (Bradford, 2016)

Confidentiality is ideally achieved by encryption Xiang and Zhu (2019). In this way information is encoded and can only be accessed or decrypted by a user with the correct encryption key. By encrypting code one ensures that only individuals who have the key can view the information. In most instances the only level of encryption covered by organisations is the access to the code is through encrypted channels (this means employing SSL/TLS). Occasionally code repositories will be centrally stored on encrypted drives, but that only provides a very slight of protection overall, with 100% protection against very specific attack vectors for example, where the disks are physically stolen.

An alternative means of ensuring *Confidentiality* Xiang and Zhu (2019) is by enforcing file permissions and access control lists to restrict access to sensitive information. An in-depth discussion on restricting access to information by enforcing file permissions and access control lists follows in section 2.3.

2.1.2 Integrity

The goal of maintaining the *Integrity* of data is threefold (Breithaupt and Merkow, 2014):

1. Prevent unauthorised users from making modifications to data or programs
2. Prevent authorised users from making improper or unauthorized modifications
3. Maintain internal and external consistency of data and programs

An example of *Integrity* checking is balancing a batch of transactions to make sure that all the information is present and accurately accounted for. Balancing transactions is the act of ensuring that both sides of a balance sheet have the same total value. When a company buys something (say hardware) their cash on hand decreases but there should be an entry on the other side that shows that the value of hardware increased. In this way system data is adequately protected from accidental and intentional changes.

2.1.3 Availability

The *Availability* aspect of the triad is one of the focal points of this research. *Availability* speaks to providing authorised individuals suitable access in relation to their job function.

> "Service providers should understand that bottlenecks aren't the only threats to availability. The dreaded DDoS attack is a technique cyber villains commonly use to deny access to network resources" (Bradford, 2016)

Achieving a meaningful balance of the three components of the triad presents a challenge for organisations but there are basic principles to which organisations should subscribe. According to Breithaupt and Merkow (2014) *Confidentiality*, *Integrity*, and/or *Availability* can be protected by implementing the following principles:

- granting access only to authorized personnel
- applying encryption to information that will be sent over the Internet or stored on digital media
- periodically testing computer system security to uncover new vulnerabilities
- building software defensively, and developing a disaster recovery plan to ensure that the business can continue to exist in the event of a disaster or loss of access by personnel

2.2 The Developer's World

In order to understand how balance can be provided to the triad of CIA in the context of software development, first an understanding of that context needs to be obtained. Developers are a special class of knowledge workers—individuals "whose principal work activities require the handling and processing of information" (Heery and Noon, 2017), as will be explored in this section.

2.2.1 Writing Secure Code

While the concern of this work is with ensuring that the developer environment is secured, there is a debate as to what secure development means. Secure development is most often described as the process of developing, i.e., writing, code which has security built-in. While this notion is correct, it excludes meeting the need to secure the very environment in which the developer works. Secure development can thus refer either to the code itself or the environment. A brief discussion on the aforementioned, the writing of secure code, follows.

There is no shortage of research dedicated to the writing of better and more secure code itself. There are already systems which address the problem of ensuring that code committed back to the central repository, and that gets shipped to clients, has its *Integrity* assured (covering this component of the CIA triad). In recent years software development teams have shifted their focus to a more proactive approach to secure software engineering. The modern day software engineering process involves a host of security-focused activities and steps which include:

- threat modelling;
- static code review;
- unit testing; and
- integration testing

This only covers a few mentioned by Bodden, Payer, and Athanasopoulos (2017).

The OWASP Foundation, which is the foundation that manages the Open Web Application Security Project (OWASP) has as its core purpose to "Be the thriving global community that drives visibility and evolution in the safety and security of the world's software."[1] Their founder and 2012 OWASP Global Board Chair Williams (2010) suggests rolling security focused code review in the OWASP Code Review Introduction, and Graff and van Wyk (2003) go further and reiterate that it is important to test software at every stage of the development cycle. This includes combining both functional and non-functional testing with security in mind (Chess and West, 2007).

Automated security testing, as suggested by Hsu (2019) ensures that this can be done as part of continuous integration and build-time penetration testing. Meucci and Muller (2013) suggests tools like OWASP's Zed Attack Proxy could be used; and the Abrahamsson, Salo, Ronkainen, and Warsta (2017) suggestion, Gauntlt, makes this technically possible.

Examples of the current tools available to developers to test their code to ensure that it is secure are:

- OWASP's Zed Attack Proxy (ZAP)[2], which fully extensible and easy to use. ZAP is also an open-source project which is actively worked on by the community. (Pruteanu, 2019)
- Gauntlt[3], which provides hooks to a variety of security tools and puts them within reach of security, development and operation teams to collaborate to build rugged software.

2.2.2 Version Control Systems

Git is the most favoured software Version Control System (VCS) according to Metz (2015) with cloud offerings like GitHub[4] being most popular. It is widely used across organisations such as Microsoft, Linux and even in government[5]. GitHub continues to gain popularity because it is a collaborative repository, publicly accessible and geared towards helping open source projects collaborate on code. In 2016 Git was named the most popular VCS, with Subversion (SVN) and Mercurial both on the decline[6].

[1] https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project
[2] http://www.zaproxy.org/
[3] http://gauntlt.org/
[4] https://octoverse.github.com/
[5] https://government.github.com/community/
[6] https://rhodecode.com/insights/version-control-systems-2016

Selecting Git as the version control system to examine was a logical choice as it is described as "...one of the most popular types of Distributed Version Control System ..." (Santacroce, Olsson, Voss, and Narebski, 2016). It allows multiple developers to work on the same project simultaneously. Git employs a common cryptographic hash function to identify and name items within a repository. Furthermore, Git takes snapshots of files, which are then hashed using the SHA-1 cryptographic hash value for each file — incidentally supporting the third leg of the CIA triad (*Integrity*) quite well. If so much as one bit is changed the next version of the same file's cryptographic hash will be changed. If a file is unchanged Git simply links to the previous files (Loeliger and McCullough, 2012). This makes it more difficult to destructively lose large amounts of work by distributing multiple copies of the history to each developer workstation. Another advantage is that changes can be implemented easily through means of merges of two developers' efforts by looking at the `diff` between their work.

The benefits identified for making use of Git for developers (Chacon, 2009) are:

- Feature branch workflow creates an isolated environment for every code change.
- Distributed development gives each developer their own local repository instead of a working copy — which is the case with SVN. This also means that one does not need a network connection to create commits or review files.
- Enable pull requests with code management tools, which means branches can be merged.
- All of the above results in a faster release cycle, which is a major advantage for business.

2.2.3 Security Policies and the Developer

Developers are wary of security policies as they view the policies as obstacles and restrictions to deploy code (Verdon, 2006). While most companies will have various security related policies such as a corporate security policy, email policy, privacy policy and so forth, it is the information systems security policy that hinders developers most. This policy reins the developer in by forcing them to comply with very specific procedures such as minimum baseline standards for server operating system security that might cause regression-testing issues with a web application. As pointed out by Verdon (2006), this issue could be overcome by having a risk exemption policy in place. An exemption policy's value is that it recognises that policies in general are 'one size fits all' and that in some instances following the policy could be counterproductive to an organisation's bottom line.

> "Typically, an exemption process is appropriate when following the policies leads to an intractable problem in deploying the application, or when the actual risk doesn't warrant the cost of following the policy." (Verdon, 2006).

A sound policy that will actually be effective and not an obstacle to a developer needs to be malleable to some degree. It cannot be tied to specific technology, which is bound to change. All policies need regular updating in order to remain relevant. The context of the code being written needs to be taken into account, as not all code deployment poses equal measures of risk (Verdon, 2006).

2.2.4 Restricting the Developer's Administrative Rights

Limiting the administrative rights of the developer seems like a logical solution to tightening development security. However, it over-simplifies the problem. Much of what developers do, such as installing drivers or writing to system directories requires the very administrative rights which they do not have. While the days of giving everyone involved in a project administrative rights are long over, implementing a system that is practical, and which provides developers enough room to do their jobs effectively, is necessary — without foregoing security all together.

Grimes (2012) provides an interesting solution to meeting developers half way by giving them the necessary freedom while not compromising security. He suggests that a developer

is issued two machines: one which has full administrative rights and one which does not. The full administrative rights box should be locked down, network shunted and only have access to necessary resources. Alternatively, developers should only be full administrators on a local virtual instance which has no access to the network or the internet (Grimes, 2014). While these suggestions are worth considering, they may not be a real solution since many development houses will not want to incur the costs of issuing each developer with two machines. Furthermore, developers rely on internet access in order to troubleshoot bugs as fast as possible.

The bottom line is that, stereotypically, developers loathe restrictions, especially those that are seen as bureaucratic and prescriptive — applied without thinking through the effects. While security is rightly a priority for an organisation, keeping a key asset — in this case the individual hired to write code — satisfied is equally important. There are a number of anecdotal accounts of developers voluntarily leaving jobs where they feel the restrictions are too cumbersome and hinder them from performing well[7]. Although there are very few academic studies to back this up it does appear to be widely accepted as fact. Graziotin, Fagerholm, Wang, and Abrahamsson (2017) did find a direct correlation between developer unhappiness and *low productivity* due to external factors (something outside of the developer themselves causing the low productivity) as well as between *delay* (again externally caused) and that same unhappiness.

2.2.5 The Consequences of a Breach of Code

The software development industry is an integral part of the larger industry of Information Technology. Software systems are complex and intangible. Davidoff (2019) explains that compromising source code can lead to larger attacks, the code can be released as open source impacting the company from which is stolen financially, it can be used to disseminate a copycat version of the same code with added malware embedded in it. In terms of the malicious insider, the stolen code can be sold to the developer's competitor and this can give them an unfair advantage. The fact remains that in the development industry source code is analogous with a high stakes golden ticket. Therefore, the dire consequences of source code being leaked, lost or stolen cannot underestimated.

As recently as June 2017, Microsoft had a portion of its source code for Windows 10 leaked (Warren, 2017). Although the leak is considered minor, the consequences of having

[7] https://workplace.stackexchange.com/questions/35893/as-a-developer-how-can-i-ask-for-more-freedom-when-confronted-with-a-tight-it-s

your company's code lifted could have a far reaching effect. The possibility exists that it could be used to identify vulnerabilities within the organisation or application for larger malicious attacks in future. Merely the whiff of that led to huge amounts of speculation at the Microsoft leak around what that code might have contained and how much it might reveal to possible attackers.

2.2.6 Separation of Powers in IT to Aid Security

Separation of duties and powers is a rational approach to prevent conflicts of interest and fraud. In the software development environment an implementation of a separation of powers means that developers are never given administrative rights on the production side (Gregg, Nam, Northcutt, and Pokladnik, 2017). A valid criticism against this angle of security is that DevOps (Development / Operations) and other core Agile development practices, as is the preferred development practice in an increasing number of organisations, require that a developer have access to the production arena so that issues can be easily and swiftly remedied. The lines between development and production are increasingly blurred when an agile development route is chosen above waterfall, for instance.

Many institutions, especially in the world of finance, still favour waterfall development methodology over an agile one, based on the perceived security related risks. In those environments Change Control, as recommended by Kissel, Stine, Scholl, Rossman, Fahlsing, and Gulick (2008) in the National Institute of Standards and Technology (NIST) Security Considerations in the System Development Life Cycle, is usually implemented along with a separation of powers. Separation of Powers is described as the principle that the completion of any task containing confidential or sensitive data require at least two people (Whitman and Mattord, 2014).

Should the separation of powers truly impact an organisation's security positively, then Gregg *et al.* (2017) suggest a number of points, including the separation of categories of staff out into separate virtual local area networks (VLANs) and the use of demilitarized zones (DMZs). Some other suggestions include the following:

- Software developers should never be given access to production environments and no code should be written in production. Furthermore, a strong change control policy needs to be put in place and strictly adhered to.

- Source code, or other intellectual property repositories, should always have a monitoring counter to detect excessive use and there should be alerts when repositories are accessed in a way that is outside of the norm.
- Backups should be made, but not everyone should have the privilege to make backups as that provides them access to a copy of the source code.
- Use unique VLAN and DMZ to restrict staff who are contracted in to work, along with strict follow through on the company's security policies.
- Because network administrators can see everything transmitted over the network they should only be allowed to use authorised sniffers, with authorised signatures and no portable sniffers should be permitted.
- Database administrators (DBA) should only have DBA authority and not root or administrator.
- Generic administrator accounts should be disabled and there should be alerts if they are used.
- Logging for systems should be directed to a write-only logging system. The system should be administered by a group separate from the individuals responsible for network and systems administration. Furthermore, access to the logging system should be role-based so that administrators may only see the logs for their own systems.
- The CIO or other officer responsible for roll-out should not have signature authority over security or compliance workers or tasks.

Using unique VLANs for different types of traffic is a proven practice and by using network zones to separate contractor's machines — that are not managed — from the rest of the managed network, one can ensure that they are unlikely to be the source of (for example) a malware outbreak. Using a DMZ to allow those machines to access resources that are otherwise protected further improves the security posture.

2.2.7 Mitigating Risks in Virtual Environments

It has been suggested that making virtual machines available to developers to work on adequately secures the developer's environment. The 2013 Cloud Computing Top Threats

report — which covers cloud and virtualisation threats including those in private clouds — by the CSA[8] clearly indicates that this is not the case. The following were identified as critical threats (ranked in order of severity) (Chaudhuri, Ferrer, Prafullchandra, Sherry, Ng, Xiaoyu, Yao Sing, and Yiak Por, 2015):

1. Data breaches
2. Data loss
3. Account or service traffic hijacking
4. Insecure interfaces and APIs (application programming interface)
5. Denial of services
6. Malicious insiders
7. Abuse of cloud services
8. Insufficient due diligence
9. Shared technology vulnerabilities

For all intents and purposes, Virtual Machines (VMs) should be treated as physical machines.

Security risks and concerns around virtual IT systems can be broadly classified into three types, namely: architectural, hypervisor software and configuration (Chaudhuri *et al.*, 2015).

In terms of architecture, the layer of abstraction between physical hardware and virtualised systems running IT services is a potential target for attack. A VM or group of VMs connected to the same network can be the target of attacks from other VMs on the network.

Hypervisor software is the most important software in a virtual IT system. Any security vulnerability in the hypervisor, associated infrastructure and management software / tools puts VMs at risk.

[8] https://cloudsecurityalliance.org/group/top-threats/

Considering configuration, given the ease of cloning and copying images in a virtual environment, a new infrastructure can be deployed very easily. This can introduce configuration drift. As a result, controlling and accounting for rapidly deployed environments becomes a critical task and tools for this purpose have been developed. There is a growing move towards "infrastructure as code", which is the practice of defining configuration in a codified way using languages (like YAML[9] or a domain specific language) and tools (like Chef[10], Puppet[11] and Ansible[12]) developed for this purpose.

Enterprises need to identify, assess and address potential security risks in the virtual environment before implementing VMs. According to Chaudhuri *et al.* (2015), high on the agenda of risks in the virtual environment are the following: lack of control processes to manage the VM lifecycle; lack of policies or enforcement of where a dormant VM can be instantiated or reside; and no discovery tools to identify unauthorized VMs. Many of these issues are resolvable but require additional controls to be put in place.

2.3 Network Access Control (NAC)

The problem of providing developers access to source code in a controlled manner has a physical world analogue: providing devices such as personal or work computers with access to the network.

There are a number of solutions which attempt to solve that physical problem but Network Access Control (NAC) is the one most commonly implemented by organisations. NAC is the automated mechanism of verifying a node's eligibility for connecting to a network and being able to access various systems by scoring the node based on a set of criteria defined in advance.

NAC makes use of Access Control Lists (ACLs) as a means of securing the network in an organisation (Lane, Conklin, White, and Williams, 2019) or for controlling what users (and applications on behalf of those users) can do on systems (Sandhu and Samarati, 1994). Essentially ACLs are the means by which a NAC solution determines access permissions.

Bogdanovic, Sreenivasa, Huang, and Blair (2015) aptly describe an ACL as an ordered set of rules that are used to filter traffic on a networking device. In the given context,

[9] https://yaml.org/
[10] https://www.chef.io/
[11] https://puppet.com/
[12] https://www.ansible.com/

each rule is represented by an Access Control Entry (ACE) and each ACE has a group of match criteria and a group of action criteria. The match criteria consists of a tuple of packet header match criteria which could further include metadata match criteria.

- Packet header matches apply to fields visible in the packet, i.e., address, class of service or port numbers
- Given a vendor supports it, metadata matches apply to fields associated with the packet — but not in the packet header such as input interface or overall packet length

The actions specify what to do with the packet when the matching criteria is met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding. The list of potential actions is endless depending on the innovations of the networked devices (Bogdanovic *et al.*, 2015). For this reason, NAC can be tailored to meet the needs of an organisation while ACLs provide specific layers of control on the network.

Implementing ACL achieves the following (Qian, Hinrichs, and Nahrstedt, 2001):

- It limits network traffic and thereby increases network performance
- It controls traffic flow
- It serves as a basic level of security for network access by defining which part of the network/server/service can be accessed by a host and which cannot
- It provides granular control over traffic entering or exiting the network

The value of ACLs becomes apparent to an organisation's security when it is implemented in a way that provides real time change notification and comment on changes to substantiate the existing rules. It then allows an organisation to configure access properly as per the business rules required and then to monitor changes to those rules as they happen. ACLs also become a significant tool in solving security issues when they can be used in audit trails and forensic analysis (Vasbinder, 2003).

Unix / NTFS Access Control Lists

Access Control Lists (ACL) are also in use as the mechanism for granting (or refusing) access to file systems sources in both the Unix (Linux / OSX) and Windows platforms respectively. Files and directories can have access granted to them based on individual access granted as well as group membership.

An ACL is an ordered list of Access Control Entries (ACEs). Each ACE is generically made up of the following access control items:

- A SID (Security Identifier) that identifies a particular user or group.
- An access mask that specifies access rights.
- A set of bit flags which determine whether or not child objects can inherit the ACE.
- A flag that indicates the type of ACE.

ACEs can be explicit or inherited and can be applied to groups (and therefore all members of the group) through the group identifier (GID) or to individual user accounts through the user identifier (UID). The order in which ACEs are applied on Windows NTFS according to Schedl (2019) is first "explicit AECs" are applied and then "inherited ACEs" are with access denied before granted (see figure 2.1).

Because Access Control Lists are an effective way to allow and deny traffic through a device to grant or deny access to file system resources, they are often used in Network Access Control systems to ensure that devices are granted or denied access to resources on the network and this is also why they are used on file systems. However, ACLs only serve as a thin veneer of security. The caveat lies in the nature of the lists: they are binary. Simply always granting access in this way makes for a poor security model for developer-specific access to source code as developers themselves are the very people who would automatically be granted access in order to do their jobs, this effectively renders them useless for the purpose of this work.

Shapland (2015) describes Network Access Control as a method of securing a network by denying access to devices: "unless they meet a predefined business policy, which are enforced by network access control products." This implies that there is an entity on the endpoint that is queried to determine the status of compliance with that business policy.

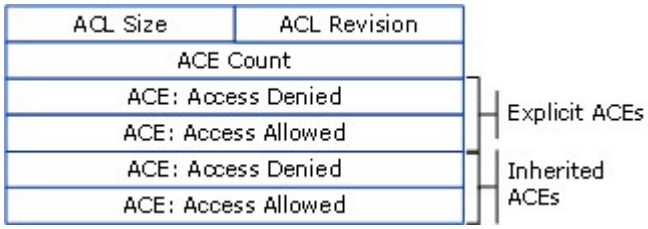


Figure 2.1: Access Control Entries (Schedl, 2019)

Network Access Control is essentially a means of preventing unauthorized devices from accessing resources on the network. It also ensures compliance with certain standards before granting access to a given system or set of systems. Those standards are guided by best practices and by the same token reflect the appetite for risk that the owners of the network have. For example, it was standard practise according to Hardy (2013) to deny unknown devices access to the network completely. However, with the advent of Bring Your Own Devices (BYOD), some companies may choose to allow personal devices in the enterprise and would then modify those rules accordingly. Shapland (2015) describes connecting NAC systems with mobile device management (MDM) offerings in order to allow for instances where developers are working from their personal machines.

The policies which NAC systems enforce are elected to complement the system and ensuring compliance with those rules is a technical challenge presently solved. This thesis investigates two commercial implementations of NAC and reviews how those distinct implementations approach the intricacy of trust together with the verification of validity of that trust in section 6.1.1.

2.4 Data Loss Prevention (DLP)

Data Loss Prevention (DLP) solutions are designed to provide organisations with peace of mind with regards to the loss, or potential loss, of all manner of intellectual property. Some even offer specific capabilities to protect against data loss in the form of source code.

DLP is one of the most prevalent mechanisms of control that an organisation has at its disposal with regards to the flow of their structured data. It is also a preventative control which organisations apply to prevent data and sensitive information from being leaked from within their organisation. There are many definitions of what DLP solutions are, the definition used by (Mogull, n.d) states that a DLP solution can be defined as a solution

that, based on central policies, identifies, monitors, and protects data — at rest, in motion, and in use — through deep content analysis. DLP solutions are both for the protection of sensitive data and to provide better insight into how the data is used and in which context it should be used. Most organisations take the approach of classifying their data as either public or private. The success of a DLP system depends on how it is configured and how the data is structured. DLP systems generally work by using deep content analysis which has the ability to examine the contents of a file to determine whether it meets the criteria of structured data which is identifiable as sensitive or confidential. These frameworks are designed to detect and prevent the unauthorized transmission of confidential or sensitive information.

When the situation entails losing data or having data leaked there are a number of areas which need to be considered. The first is the states in which the data exists. Within any organisation there will be data which is in motion, meaning that is moving across the internal network either to the external world, or simply to another host on the same network. Then there is data which is at rest, i.e., data that resides in the file systems of the host, in databases, or on other storage devices (such as block storage). The data can also be contained on the physical end-user controlled and mobile, endpoints on the network such as:

- Universal Serial Bus (USB) flash drives;
- external hard drives;
- laptop computers; and even
- mobile devices

In the context of the research which hopes to find a better way to protect the development pipeline and intellectual property, organisations are faced with the reality of protecting their secrets. Often with security in any industry, the balance between *Confidentiality*, *Availability* and *Integrity* is skewed towards one (or two) point(s).

Some DLP systems aid in protecting data leakage of source code in that they have the ability to block developers from copying source code into emails, transferring it via social media channels or otherwise ex-filtrating on the internet. Several DLP systems take it a step further in that they can stop the source code from being downloaded onto an external device. The libraries which DLP solutions make use of often utilize complex structures to

be in a position to identify various programming languages. Not all DLP solutions offer the ability to distinguish between programming languages.

Through this work three solutions, that provide specific source code protection as part of the solution, will be examined in terms of their offerings:

1. MyDLP[13] offers an information type to look for source code (Ozgen, 2018) that can be added to any rules covering data transfer;
2. EndPoint Protector[14] offers N-gram-based text categorisation (Coos, 2018) to identify (and subsequently protect) source code; and
3. Symantec[15] offers a four step program (Fitzgerald, 2016) to provide source code protection using their DLP products.

2.4.1 MyDLP

The MyDLP solution is specifically designed for data leakage prevention solutions that offer network protection, endpoint protection and discovery. When using MyDLP an organisation has the ability to monitor and prevent confidential data from leaving the organisation via the network. It also has the means to monitor for external removable media being used to copy confidential and sensitive data. The solution additionally provides the capability to monitor smart phones as well as printing jobs which contain confidential media and stop these jobs from being printed. The MyDLP solution can scan and discover confidential data on workstations and laptops within the organisation. Its ability to identify data extends to:

- Keywords
- Regular Expressions
- Partial and approximate document matching
- Credit Card Numbers
- International Bank Account Numbers (IBAN)

[13] https://www.mydlp.com/why-mydlp/
[14] https://www.endpointprotector.com/blog/keep-source-code-safe-with-dlp/
[15] https://www.symantec.com/products/information-centric-security

- Social Security and identification numbers
- Source code identification

Unlike some of the other solutions discussed in this paper, this particular DLP solution only has the ability to identify source code written in Ada (within a data stream or file) as well as C, C++ C# and Java. In relation to the problem statement in section 1.2, this solution focuses only on the identification of the movement of data which has been classified as secret or confidential, and does not provide a balancing mechanism. In this instance, if source code is marked as such (or is identified as such) then it does offer more peace of mind to an organisation in that they know what the movements are with regards to their source code. This would allow an organisation to prohibit developers from being able to copy repositories to their local device. Subsequently, developers would need to work on only a remote repository rather than work locally reducing the *Availability* of the source code to them alone.

While it does provide a method to control the movement of the source code to battle the malicious developer, it can have impacts on the productivity of all other developers who are in the same boat. The importance for a developer to have local access to repositories is a critical element to code production and is discussed in detail in section 2.2.

However, the most far-reaching downfall of using MyDLP is that it is reliant on string detection. String detection can be described as finding occurrences of a pattern string within another string or body of text. The use of string matching has been an efficient tool in the detection of plagiarism of code within software development according to Pandey, Agarwal, Misra, and Prasad (2012), Lee, Lim, Ji, Cho, and Woo (2012), and more recently Novak, Joy, and Kermek (2019). Complex strings can be missed which means that there will be notifications of apparent attempts to ex-filtrate confidential data that turn out to be false positives, for example, a developer may use a pattern of development that is common to the team to answer a question on StackOverflow[16] such a piece of code might be highlighted as an attempt to ex-filtrate confidential source code when it is not.

This is not a comprehensive solution to the problem statement which is focused on balancing *Confidentiality* with *Availability* as the sole focus here is on the *Confidentiality* requirements of the organisation. It effectively, and substantially, detracts from the *Availability* of the source code locally from the developer by placing limits on how (and where) they may access it from.

[16] https://stackoverflow.com/

2.4.2 EndPoint Protector

EndPoint Protector is specifically designed to protect confidential data against the insider threat while not impacting on productivity. It follows a white-list and blacklist approach which grants flexibility in terms of policy building. This resolution gives an organisation the ability to limit or prohibit the use of specific removable devices or data transfers. It can be deployed as either a hardware or virtual appliance. The solution is content aware, that is to say that if data leaves the endpoint through cloud applications, USB sticks or e-mail, the organisation needs assurance that the content is not against their policy. Additionally, it can run on multiple operating system platforms, namely Windows, macOS and Linux. The key difference it has from MyDLP is that it enforces encryption on the Windows and macOS operating systems which means that only authorized USB devices can copy data onto a removable disk. In addition, it also manages encrypting and securing the data automatically.

Content aware protection allows a deployment to have versatility in how these content conscious policies are set up in order to prevent any files which match source code file patterns being moved outside of the network.

EndPoint Protector also permits the setup of different hierarchies and groupings for endpoint management in relation to access to source code.

The solution invites integration with and the monitoring of GitHub (which is often used in production environments to host repositories). EndPoint Protector makes use of N-Gram-Based text categorization. It is a contiguous sequence of n items from a given sample of text or speech. This is a probability language model which predicts what the next item in a certain sequence would most likely be. It is specifically used because it is simplistic and can be scaled. In code development, strings are not always simple in nature and can contain complex computation elements. For this reason, identifying the language is complicated.

Data *Confidentiality*, *Integrity* and *Availability* are the quintessential goals for all information security mechanisms. However, the difficulty with most security instruments that that they do not proactively protect sensitive data. More readily, they work with specific predefined policies and conditions which authorises them to identify possible instances where the policies were breached. In the research done by Alneyadi, Sithirasenan, and Muthukkumarasamy (2014) an examination of the statistical analysis of data fingerprinting of N-grams was performed. N-gram classification relies on the frequency of keywords

found to classify documents under distinct categories. N-grams statistical analysis is done by breaking down a document into single word N-grams. These are then sorted from the most frequent to the least frequent. The final step is to remove all stop words, common words and phrases.

While the research mostly covered general text and not only source code, Alneyadi *et al.* (2014), did briefly look at N-grams for source. Their research indicates that in their findings the effectiveness of N-gram model of detection was influenced by the size of the documents. The smaller the document, the more accurate the detection. Conversely, in the case of larger files the detection was less accurate. However, depending on the classification model and the data, it was established that N-gram detection could have a success rate of as high as 90.56% in fairly ideal conditions. While source code documents tend be more extensive, they are also more structured. The former would force one to manage more and more N-grams. This, in turn, forces the detection success rate off the more optimal testing. Consequently, it is less accurate if the same principles apply as those with the researches (Alneyadi *et al.*, 2014).

In terms of the problem description discussed in section 1.2, the Endpoint solution offers a more balanced resolution than MyDLP. Its ability to allow for data to be remotely kept as long as it meets the requirements as set out in the policy, is a major advantage. Furthermore, it offers to encrypt this data on location. Nevertheless, Endpoint is not without its shortcomings. With respect to the development pipeline, the solution leans more towards the *Confidentiality* aspect in protecting the source code than allowing free access to it. It also does not offer local support, as files which are kept locally are read-only. Subsequently, the developer would need to be on the network and access the files over the network without having the necessary ability to keep making changes to the local repository. This DLP solution will need to be accurately set up to tag relevant data streams within the source code and anticipate what the complex strings may be. This will likely lead to false positives or the failure to spot relevant data movement.

2.4.3 Symantec

Symantec Data Loss Prevention (SDLP), which has been designed for endpoint protection, addresses the risks associated with the storage and use of confidential data on laptops and desktops spread across an organisation. This DLP solution for endpoints discovers confidential data wherever it resides and identifies the endpoints which are most at risk.

It has the ability to actively monitor endpoints for activity and data movement. This one solution is made up of effectively two different solutions:

- SDLP EndPoint Discover; and
- SDLP Prevent.

Making use of these two products concurrently affords an organisation the capability to scan endpoints for confidential data in order to create a type of inventory to secure and relocate it. Said inventory is initiated when the devices are idle for the purpose of setting a baseline. It then uses incremental scans to determine any variances since the most recent scan. The detection model will seek out data which matches specific criteria i.e. keywords or expressions as well as signature based detection. It uses fingerprinting to search for exact matches to whole or partial files — including unstructured data fingerprinted via hashing algorithms. It also creates an opportunity for some level of learning to be able to look for unstructured data such as source code or intellectual property. This is performed based on a statistical model created by uploading positive and negative examples of documents. The Symantec solution, like many of the other DLP solutions, is only as accurate the data which it is given to ingest. DLP systems are fundamentally built to detect the use of data which has been categorised as sensitive. Their success rests on the categorisation of data.

The second portion of the SDLP system for endpoints is the Prevent solution. It refers to the portion of the solution that is used to give extensive coverage of end user activity in a context aware manner. The implication is that the data being used is as important as the purpose for which it is needed in order to determine whether the purpose the data is currently being used for is permitted or not. The two other solutions which were reviewed, namely EndPoint Protector (section 2.4.2) and MyDLP (section 2.4.1) do not take into account what the data is used for in the same vein as this product. This would result in a developer possibly not having access to source code which is labeled confidential and it would not trigger the DLP alerts or other actions conditional to the way the data was employed. An example of data sent to GitHub illustrates the point in question. A number of DLP solutions would mark and block the action as it would be construed the as transference of confidential data to cloud storage. Be that as it may, this solution would take into account the particular data being sent in addition to the context. The solution further allows devices to be defined specifically with regards to which devices can be used and what confidential data can be placed on them. This will be particularly beneficial as it gives developers access to locally downloaded repositories.

SDLP has a component that does application file access control. It secures the access that applications such as social media platforms have to your confidential files. This can also be used to limit the information that can be accessed by the development platform used by the developers.

SDLP offers an effective way to provide developers access to the data in the right context required. This meets the need for *Availability* of confidential data portion of the problem statement. The solution's weakness can be identified in that it has no protection enabled: once the local repository is downloaded, the data is resident on the local machine. Because once there it's not encrypted, additional alternative solutions are necessary to make the *Confidentiality* aspect of the problem statement more comprehensive.

2.5 Google BeyondCorp Implementations

Google BeyondCorp is a concept that Google has proffered in the zero-trust networking realm. It can be described as a piecemeal approach to the implementation of several concepts that have been deliberated as the best practices. Those best practices include end-to-end encryption, service isolation and least privilege access to applications.

The name BeyondCorp indicates the stated goal of the program i.e. to remove the concept of a corporate network and instead allow employees to access systems previous only accessible from that network over the public internet.

Requirements

The requirements for BeyondCorp to work are:

- Device identity;
- User identity; and
- The ability to infer trust

The ability to positively identify both a device and user are critical to establishing trust. There are a number of possible solutions for each. Essentially, however, the method used needs to be reliable and needs to have a very low level of dependence on other

infrastructure components. Once one can identify the user and their device, one needs the ability to establish the state of those two components.

With respect to the user, this is generally through a central management system which tracks users, their active status, and the groups of which they are members. Often this is a Lightweight Directory Access Protocol (LDAP). LDAP are solutions in the same vein as Azure AD, Active Directory, or an OAuth solution. These approaches allow for users to be made members of groups that are related to the access they are assigned. This function can be enabled or disabled as required.

For the device in question this process can lend itself to certain complications. As such, the level of granularity of details about the device deemed to be necessary for the type of access the device may require, can differ. For example, one may only want to allow corporate devices to access a service, and those devices may need to have full disk encryption enabled and a particular level of patching enabled. The tooling required to provide that information will need to be in place in order to progress to the final part.

Trust inference is the term used to describe the ability to combine all the information required (as described previously) to make a decision regarding the state of the user and device when they attempt to access the system in question.

Google's BeyondCorp implementation of a zero-trust computing environment leverages several components that bear a strong resemblance to the architecture proposed in this work as it takes advantage of several components discussed throughout this work.

Components

The BeyondCorp solution is made up of a number of components (see Figure 2.2), namely:

- the access control engine (ACE);
- the application proxy (AP);
- a pipeline; and
- the system being protected

The role of the access control engine (ACE) is to determine, based on the access control policy (ACP) specified for the service being accessed, what the requirements for access

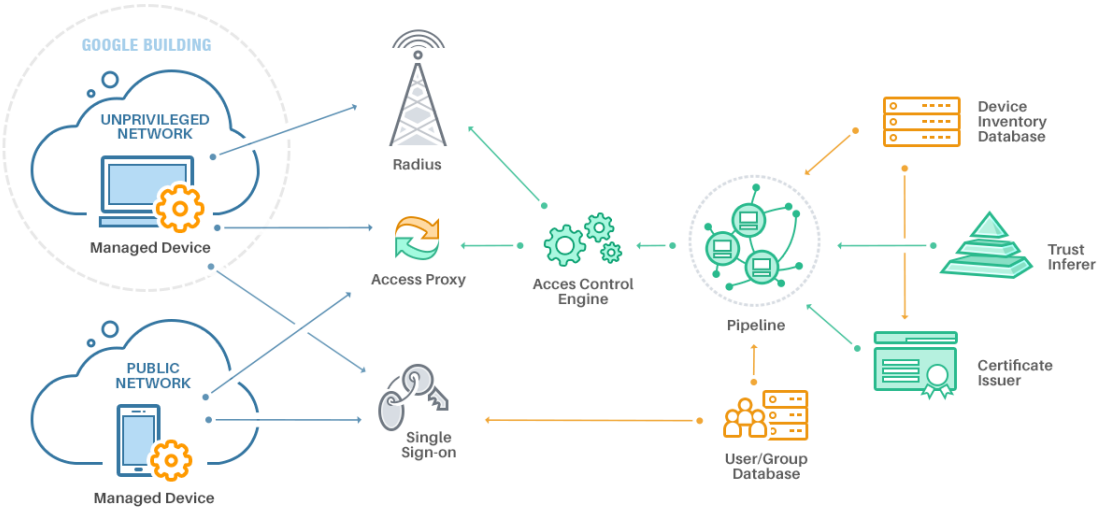


Figure 2.2: Google BeyondCorp — Reference Architecture (Dwyer, 2017)

are. For example: if this is a service that will push data out to the end point device, then the ACE will need to determine whether the device supports full disk encryption and if it is enabled.

The role of the Application Proxy is to facilitate access to the service by presenting it over for example Transport Layer Security (TLS) or Secure Shell (SSH) to the users of that service. These two specific protocols are generally used because they are encrypted channels widely used and supported by multiple services on the internet.

2.6 Git-related Solutions

There have been a number of partial solutions to address the concerns of Confidentiality for source code repositories that have come out of the Git world themselves. Development of systems and solutions to improve Git's ability to keep secrets have focused on encryption in three primary ways.

2.6.1 Transparent Git Encryption

The need for transparent encryption may arise when working with a remote Git repository which is hosted on a third-party storage server in order to secure confidential information. A good example of this is Dropbox. Dropbox is a widely accepted storage tool in the

corporate sphere despite being a serious and high security risk as it relies on third party storage (Shang, 2011).

Rota (2014) explains the workings of the encrypted Git as follows:

> "Files which you choose to protect are encrypted when committed, and decrypted when checked out. `git-crypt` lets you freely share a repository containing a mix of public and private content...it gracefully degrades, so developers without the secret key can still clone and commit to a repository with encrypted files. This lets you store your secret material (such as keys or passwords) in the same repository as your code, without requiring you to lock down your entire repository."

Before data is pushed to the remote repository, it is encrypted with an encryption key which is only known to the owner of the data (Shang, 2011). Once the end-to-end encryption is set up, one can work in the local repository and push/pull to and from the remote repository as usual, without noticing the encryption/decryption in the background (Shang, 2011).

It must be added that although Transparent Git Encryption protects the data of the file, it will not encrypt the file name, which may be deemed a chink in its security armour. The encryption protects only things specifically labeled as secrets in the source code (i.e. files containing passwords or access keys) but would still allow someone who gains access to the source code to have access to everything else in the code itself.

The problem statement discussed in section 1.2 discusses the fact that developers need to be afforded access to source code, whilst assuring an organisation a measure of control. Transparent Git Encryption was examined in terms of the CIA triad to determine how it addresses the need for *Availability* of the repository to the developer versus what level of *Confidentiality* it offers to the organisation which owns the source code. Transparent Git Encryption is specifically designed to protect repositories that are stored on third party storage solutions, and the focus of encryption is to allow for data to remain secure whilst stored there. This also means that data is only encrypted while remote.

Transparent Git-encryption makes the *Availability* aspect of the whole repository to developers possible in terms of allowing them to push/pull from the remote repository. In addition, they may not even notice the encryption and decryption process which is handled on the back-end. In terms of addressing the problem statement it leans fairly towards *Availability* within the CIA triad.

The method which is utilized by this encryption allows for specific data to be labelled as secret and be encrypted with something which is known only to the owner of the data. The encryption process is fully end-to-end encryption; data is encrypted locally and transferred in that encrypted form to the remote repository. It's worth noting that the filenames of the data which are encrypted are unchanged and could potentially give away something of the purpose of the files. This does not provide *Confidentiality* of data for the local repository. Furthermore, the source code itself is not fully encrypted, only sensitive portions which have been labelled "secret" are. Generally the use case here would be to do this for passwords or access keys. This leaves the remainder of the source code exposed and available to be accessed.

This particular Git resource is more aligned to the *Availability* of the source code. It does focus on providing some form of *Confidentiality* in terms of data labelled secret. This does not protect the local repositories as the focus is on the encryption of data in transit and remote storage. The balance in terms of *Availability* and *Confidentiality* is not satisfy the ideal balance of the CIA triad and certainly it provides far too little in terms of *Confidentiality* for the local repository.

2.6.2 git-remote-gcrypt

`git-remote-gcrypt` is a Git remote helper to push and pull from repositories encrypted with GnuPG (GPG). It also operates with the standard Git transports, including repository hosting services like GitHub (Whitton, 2016).

Even with remote Git encryption, hardware theft still poses some risk to security. This is because, although `git-remote-gcrypt` is used to `git push` and `git fetch` from the client computers to the server, on the client side the source code remains unencrypted. Furthermore, the contents and the names of the files and other repository metadata is also encrypted.

Once the encryption has been set up the developer should have the ability to push to and from the repository as normal. There may be an occasional prompt to enter the `gpg` key passphrase. This can be avoided by setting up the length of time the `gpg-agent` will cache passphrases for.

`git-remote-gcrypt` requires the entire repository history to be uploaded with each push. If the repository history is substantial, and one is pushing this data over a slow link,

managing the push and pull will be slow and prone to timeouts. Incremental uploads and downloads are supported in an attempt to alleviate the issue. There are a number of unfavourable repercussions to take into account. Although Git natively supports this option it is not supported on the most popular Git solutions: Gitolite, GitHub or GitLab. There is also no compression option when using this to secure and encrypt the whole repository. This solution would not provide a seamless pull or push to and from the repository for larger repositories owing to the full history which requires sending. When the `git-remote-gcrypt` has network connectivity issues, or authentication difficulties it may report that the repository does not exist even though in fact it does exist.

`git-remote-gcrypt` was reviewed in terms of section 1.2, to examine whether this solution offers the *Availability* to the developer whilst maintaining the *Confidentiality* of the source code for the organisation which owns it. This resource affords the developer access to the repository both remotely and locally. The main constraint is that the pull and push request will not be seamless. The reason for this is that `gpg` only caches the passphrase for a limited time and the developer will need to periodically re-enter this information to gain access to the remote repository. A further constraint is that the full history of the repository will be downloaded with each pull-push attempt. This, in turn, makes the process slower than the other solutions which have been examined.

The full encryption of the repository in terms of this solution negatively influences the ease of *Availability* for the developer. Conversely, it does benefit *Confidentiality* for the organisation which owns the repository as everything is encrypted.

2.6.3 git-crypt

According to the `git-crypt` GitHub website[17] "`git-crypt` is more secure that other transparent Git encryption systems [because] it encrypts files using AES-256 in CTR mode with a synthetic IV derived from the SHA-1 HMAC of the file." (Ayer, 2017a) This mode of operation secures against the deterministic chosen-plaintext attack. This means that although the encryption is deterministic (which is required so that Git can distinguish when a file has and has not changed), it leaks no information beyond whether two files are identical or not (Ayer, 2017a). `git-crypt` allows a repository owner to select which files are to be encrypted by specifying them in a `.gitattributes` file which works much like the `.gitignore` file and can use wild cards and the like to specify files for encryption.

[17] https://github.com/AGWA/git-crypt

However, `git-crypt` has its limitations. The main issue identified by users is that the transformation between the encrypted and decrypted state is transparent, which means it's very easy to commit files in clear that should have been encrypted (zimbatm, 2014).

Other `git-crypt` security issues are:

- it relies on Git file filters (like `.gitignore` does) which were not designed for encryption (and are easily confused). The best option for encrypting a whole repository is using `git-remote-gcrypt` (covered in section 2.6.2).
- `git-crypt` does not encrypt file names, commit messages, symlink targets, `gitlinks`, or other metadata (Ayer, 2017b).
- it does not hide whether a file has or has not changed, i.e. the deterministic nature of its encryption reveals whether a file changed or not.
- Git files cannot be compressed (Ayer, 2017b).

Compression of the Git repository would no longer be as easy because `git-crypt` encrypts before compression would normally occur and the act of encryption results in padded, random text ill suited to being compressed. For this reason, compression is generally performed before encryption — which is not possible with `git-crypt`.

Based on these limitations, it's clear that `git-crypt` cannot be deemed secure unless the entire repository is protected against tampering i.e. an attacker who can mutate your repository can alter your `.gitattributes` file to disable encryption (Ayer, 2017b) as modifying that file is all that is required to enable encryption for files that match the criteria put there.

The researcher reviewed `git-crypt` in terms of how well it met the problem statement's (section 1.2) goals; in order to determine if this solution offers a balance between making source code accessible (*Availability* in the CIA terms) to developers whilst maintaining security in terms of *Confidentiality*. This solution does have the ability to encrypt the whole repository and this offers the *Confidentiality* which would be needed for an organisation, it does however need to rely on other resources to achieve this seamlessly. This solution focuses on the *Availability* for developers to access the source code, it also focuses on protecting the remote repository.

In spite of this, the data is not protected locally at rest whilst the developer is idle. This does not offer protection from physical data theft. The source code is left vulnerable at

the local point whilst protecting the the source or remote only. In terms of the research problem this does not address the *Confidentiality* of the source code to the fullest extent. Also in terms of *Availability* there are constraints in the size at which the source code will grow and limitations placed on the developer.

2.7 Summary

This chapter has discussed a spectrum of topics related to the developer's world with the purpose of determining how developers perform their jobs. It also investigated the CIA Triad to set the foundation for understanding the imbalance within software development in terms of the *Confidentiality* of and *Availability* to source code. The researcher discussed various solutions which all appear to offer only partial resolution in terms of the problem statement. It was determined that none of these solutions address the problem statement completely as discussed in chapter 2. The related work reviewed seems to indicate that more than one solution would be required to fully address the imbalance between *Confidentiality* and *Availability*. This indicates that there is not a single solution currently available that solves the imbalance of CIA triad absolutely. There is also an evident lack of research in this area to assist in this work. For this reason the focus was placed more on technical writings in other areas, on books available commercially and on the implementations of currently available solutions.

Chapter 3

Solution

Creating an unobtrusive client agent was key. That client had to ensure *Confidentiality* is improved and it also had to provide an easy way to review which code bases are accessible through it. In essence, what the researcher wants to achieve is the following: a client agent inclusive of the necessary tools to connect to Git repositories which ensures that the source code can be checked out, the storage for the source code managed, while further making the set up of development environments easier for developers. The solution developed and described here is referred to as "Code Corral". The choice of name is no accident as "corrals" are places of safety often associated with cattle (McCance, 2012)[1] and cowboys (Collins, 2012)[2].

A number of design decisions where made almost immediately, including allowing the agent to use existing authentication mechanisms (the SSH key pair that Git would traditionally have used) to control how repositories are accessed. This course of action meant that authentication becomes seamless to the developers without adding any additional friction to the adoption, and that it would not require additional work in this fairly complex area from the perspective of development. Section 3.1 discusses the design process, covering in section 3.1.1 the client side of the solution and the server aspect in section 3.1.2.

The point here was to ensure that *Availability* would be addressed by the software being as unobtrusive as possible and by default, allowing access to the source code. In section 2.2 the important and that this aspect of the triad of CIA holds was examined and the need to ensure that it remains as nonrestrictive as possible.

[1] "Future application architectures should use Cattle" on building architecture for the future and scale.
[2] "Informal. a reckless or irresponsible person, especially a show-off or one who undertakes a dangerous or sensitive task heedlessly" sometimes used to describe the best — and the worst — developers

Integrity is untouched by this experiment and remains at whatever level exists within the organisation; it should be addressed through code review, automated testing and good development practices.

The level of ***Confidentiality***, which this experiment is looking to provide a mechanism to increase, can be adjusted via rules applied to grant access. The solution provides a base set of these rules to begin with, detailed in the introduction to chapter 4 and some other possible rules can be found in Appendix C.1.

3.1 Design and Implementation

Design had to leverage existing technologies where possible so as to avoid reinventing the wheel. It also had to be as universal as possible so that it could be deployed to the diverse and dispersed developer instances found in most development environments (Herbsleb, 2007).

Code Corral consists of a client and server. The server is a web service which maintains information (stored in a database) about what clients are authorised against which projects, along with other information about those clients and projects. The client communicates with the server via APIs over Hypertext Transfer Protocol Secure (HTTPS). The client would need to communicate with the server, provide encryption and decryption services, and be able to discover information about the host it was running on. The server needed to respond to requests from the client and provide a means to authenticate a client for access to a particular project. The overview shown in figure 3.1 gives a logical view of this.

3.1.1 Client Agent

A lightweight wrapper encompassing existing open source components that would do most of the heavy lifting involved in creating the client-side environment was a prerequisite for the client agent. It also needed to be easily deployed and to run in the background with a basic command line interface (CLI) that permitted access to the features offered.

It was imperative that the feature list included the following aspects that would allow a developer to:

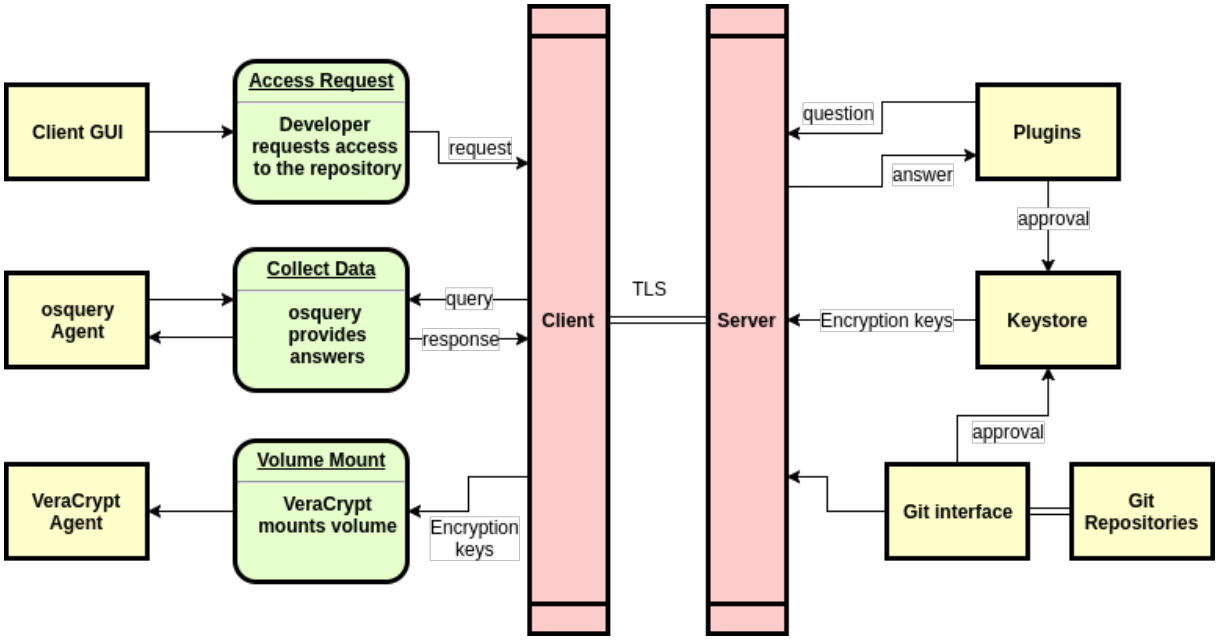


Figure 3.1: Code Corral - Logical Diagram

- configure which server instance they are connecting to; and
- to configure those repositories by providing the client with the necessary ssh public key for access to that repository

The client then required the capability to execute tests locally at the instruction of the server to determine if the developer should be granted access. For example, if the repository owner wanted only Windows machines to be allowed access, then the client needed to be able to tell if the machine it was running on was indeed, in this instance, Windows. If the repository owner wanted to ensure that a minimum level of patching had been applied, then it would be crucial that the client had the capability to query its host to determine if it met the requirements.

To ensure the most compatibility with diverse developer environments, and to limit the need for reinventing the wheel, the client side was designed as a lightweight wrapper for existing projects that provide the required functionality. The open source components to be wrapped by the client agent were ***VeraCrypt*** and ***osquery*** and the wrapper is ***bash***.

Bash Scripting

All modern operating systems make use of a type of shell subsystem. A few shells can be command line oriented and some have graphical user interfaces (GUI) interfaces. As is the

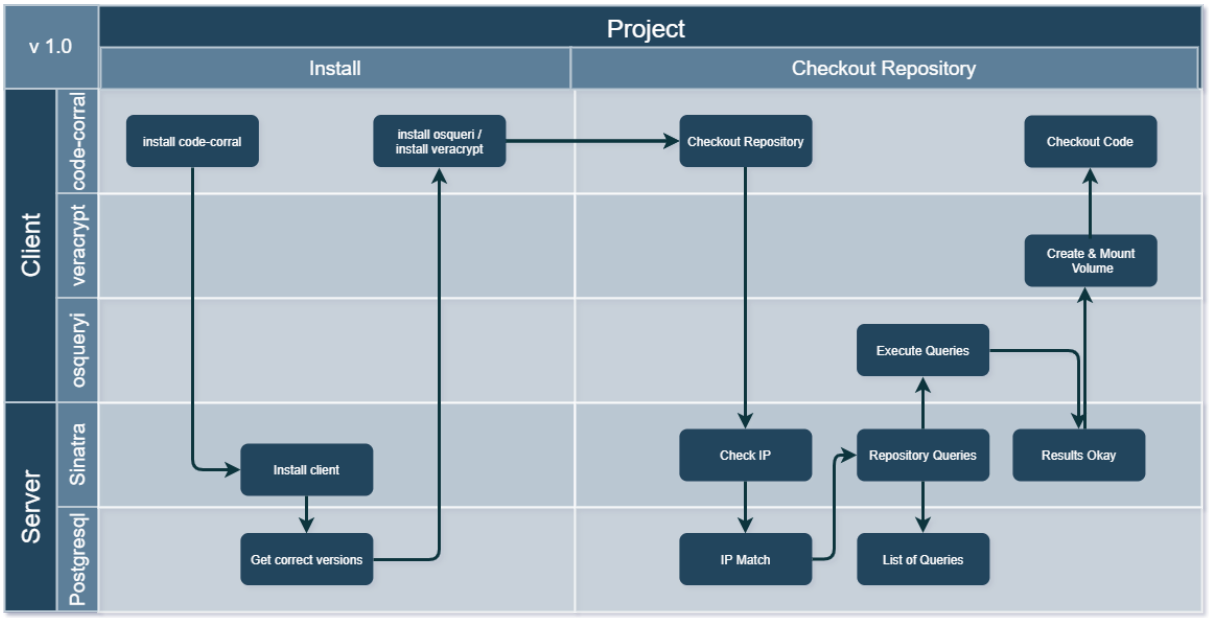


Figure 3.2: Code Corral - Checkout Process

case with most things, there is often more than one way to complete a task. Most modern operating systems offer both a user interface as well as a system interface that allows for the execution of commands. The Unix operating system popularized the idea that there needs to be a separation between the shell (user space) and other system elements (Cooper, 2014). The main purpose of the shell is to carry out commands to execute other programs on behalf of the users. That was the beginning of the shell developers know (and love) today. The modern shell as it stands presently, is convenient in that it typically has the ability to provide contextual data based on where one is on the file system and what actions have been taken. It can also remember previously used commands and also allows those to be edited (Albing, Vossen, and Newham, 2007).

All operating systems have a type of program that allows for the execution of commands. The Linux system has a terminal that usually runs bash. macOS has the same terminal (running a version of bash by default) and Windows has the command prompt or PowerShell. In Windows 10 however, the Windows Subsystem for Linux (WSL) was introduced allowing one to run full versions of Linux (and bash) on the Windows Platform. Developers, in prior versions of Windows would usually install Cygwin[3] or Git-Bash[4] to provide themselves with command line Git access and the same level of functionality. This means that, for developers, bash scripting is generally available across platforms they would be happy to run.

[3] https://cygwin.com/
[4] https://www.atlassian.com/git/tutorials/git-bash

In this case Bash not only provides the command line interface (CLI) to Code Corral, it also provides the installation mechanism (of Code Corral itself as well as all dependencies) and the setup and configuration of the client. The Code Corral CLI consists of twenty-one functions spread across seven sections with purposes ranging from setting up logging through to listing, cloning, opening, and closing projects protected by Code Corral. The significance of these functions is that they provide the main functionality of Code Corral and execute API calls to the configured Code Corral server after doing some work locally. As an example see listing 3.1 (the *Open Project* function) where the API call happens on line 28. In reality the API call is actually performed in listing 3.2, but the call initiated here on line 28. The remainder of the source code is included as listings in the appendices as well as in a public GitHub repository.

Listing 3.1: Code Corral / open_project

```
1  function open_project {
2    project_name=$1
3    clone_path=$2
4
5    if [ -z $project_name ]
6    then
7      log error "No project name provided!"
8      exit -1
9    fi
10
11   ensure_project_not_already_mounted "$project_name"
12
13   project_config_path="${CODE_CORRAL_CONFIGURATION_FOLDER}/${project_name}.
       config"
14   if [ ! -f "${project_config_path}" ]
15   then
16     log error "No configuration file found for this project"
17     exit -1
18   fi
19
20   if [ -z $clone_path ]
21   then
22     clone_path=$(cat ${project_config_path})
23   fi
24
25   mkdir -p "${clone_path}"
26   clone_path="$(cd "$(dirname "$clone_path")" && pwd)/$(basename "
       $clone_path")"
27
```

```
28   api_post "project/open/${project_name}"
29 }
```

Listing 3.2: Code Corral / make_api_call

```
1  function make_api_call {
2    request_method="$1"
3    api_call="$2"
4    data="$3"
5
6    ensure_ssh_auth_headers_are_loaded
7
8    # Execute API call
9    full_url="$CODE_CORRAL_URI/${api_call}"
10   log debug "Sending API Call : ${full_url}"
11   error_log=`mktemp`
12   curl_output=`mktemp`
13   http_status_code=$(curl --request "${request_method}" --silent --output $
       {curl_output} -w '%{http_code}' \
14     --header "X-Code-Corral-Email-Address: ${CODE_CORRAL_EMAIL_ADDRESS}" \
15     --header "X-Code-Corral-Client: ${CLIENT_SHA}" \
16     --header "X-Code-Corral-Tool-Veracrypt: ${VERACRYPT_SHA}" \
17     --header "X-Code-Corral-Tool-OSQuery: ${OSQUERY_SHA}" \
18     --header "X-Code-Corral-Auth: ${USER_AUTH_HEADER}" \
19     --data "${data}" \
20     --stderr "${error_log}" "${full_url}")
21   api_result=$(cat "${curl_output}")
22   if [[ $http_status_code -ne 200 ]]
23   then
24     log error "Failed upstream API call ${full_url} with status:
       $http_status_code $api_result" 1>&2
25     cat "${error_log}"
26     exit -1
27   fi
28   handle_api_result $api_result
29 }
```

The Bash script provides functions to install the OSQuery (`osqueryi`) and VeraCrypt `veracrypt`) binaries required for Linux. Provision was made for the same setup under Windows and macOS (for example see listing 3.3) but not implemented in the initial proof of concept (PoC) and those functions will also ensure that other Bash niceties are installed (such as curl).

Listing 3.3: Code Corral / download_and_install_veracrypt

```
function download_and_install_veracrypt {
  ensure_installation_folders
  error_log=‘mktemp‘
  tmp_binary=‘mktemp‘
  if [[ "$OSTYPE" == "darwin" ]]
  then
    os_type="osx"
  elif [[ "$OSTYPE" == "msys" ]]
  then
    os_type="win"
  else
    os_type="linux"
  fi
  full_url="$CODE_CORRAL_SERVER/veracrypt/$os_type/veracrypt"
  set +e
  curl --fail "${full_url}" 1>"$tmp_binary" 2>"${error_log}"
  curl_status_code=$?
  if [[ $curl_status_code != 0 ]]
  then
    log error "Download of ${full_url} failed with status: ${
   curl_status_code}"
    cat "${error_log}"
    exit ${curl_status_code}
  fi
  set -e
  sudo mv "$tmp_binary" "$VERACRYPT_BINARY"
  sudo chmod 755 "$VERACRYPT_BINARY"
}
```

VeraCrypt

VeraCrypt is an open source project that aims at taking the place of the very popular TrueCrypt project (Buchanan, 2018). When the TrueCrypt project was closed in May 2014, a number of projects, including VeraCrypt, were born of the ashes. Both TrueCrypt and VeraCrypt provide full disk encryption and encrypted volumes (Buchanan, 2018) but the TrueCrypt project is no longer being maintained (Buchanan, 2018) and thus is no longer a viable option for individuals and organisations who require encryption.

VeraCrypt provides a mechanism for secure storage of the cloned Git repository. It achieves this by creating an encrypted partition large enough in which the repository

can clone itself. The keys required for mounting and dismounting the partition are stored by the central server. This provide a requirement to re-authenticate, and to allow central revocation of access to the keys. The process prevents an individual with access to the volume from gaining access to valid keys.

As VeraCrypt is an open source project, a fork was created for the purposes of this project, in instances where any issues that might need correcting was discovered or modifications needed to be made. The stand-alone, so-called portable, binary for Linux was also built and included in the server deployment for the trial and testing to ensure that the installation of Code Corral (see listing 3.3) would always provide the correct, working version of VeraCrypt. The binary is downloaded using curl and stored in the same location as all the other Code Corral binaries and is executed from there.

OSQuery

OSQuery is an operating system instrumentation framework for Windows, macOS, Linux and FreeBSD. It is a low-level operating system analytical tool that operates both performantly — i.e. with little delay to the query and low impact to the client — and intuitively. The method by which this tool works is that it exposes the operating system as a high-performance relational database. This allows for the use of SQL queries to explore operating system data. When using OSQuery the tool abstracts concepts which include running processes, loaded kernel modules, open network connections, browser plugins, hardware events, and hashes of files as data that can be queried. OSQuery is made up of different elements which includes the `osqueryd` which allows for the scheduling of queries which need to be run across an entire infrastructure. The aggregation of the resulting data is done by the daemon from the results of the queries which are run. This is then logged to the OSQuery local storage (in RocksDB) where the Diff Engine can be used to indicate any changes in the infrastructure.

The interactive query console (`osqueryi`) offers a SQL interface in which new queries can be built and tested that might run with the environment. The main benefit of OSQuery is that supports multiple platforms, including the ones most of interest to this research (macOS, Windows and Linux) owing to it taking advantage of the low-level operating system APIs. This also allows one to build queries to be used across platforms to monitor, otherwise too complex, system states across client and production systems. The code base which is used by OSQuery is made up of high-performance modular components, lending it to integration with other projects.

The deployment of OSQuery across an organisation's infrastructure is seamless as it comes integrated with native packages in support of most operating systems. These components can be easily used in conjunction with each other to create new applications and tools. Language bindings exist for multiple language sets allowing one to make use of familiar and known technologies.

For the purposes of this project the OSQuery project was also forked and the portable, stand-alone version of the `osqueryi` binary for Linux was also built and included in the server deployment for installation on the client machines.

3.1.2 Server Software

PostgreSQL

PostgreSQL is an open source database management system (DBMS) which has been developed in an academic environment. The project was led by Michael Stonebraker, a professor at the University of Berkeley. PostgreSQL was one of the first relational databases to be designed. PostgreSQL is a reliable database which can be scaled to be used at enterprise-level which is advantageous. A further aspect of it is that PostgreSQL continually embeds new features, which is why applications which manage business-critical data make use of it. PostgreSQL also offers secure TLS connections that provide for various authentication methods out of the box. These include password authentication, client certificates and the ability to plug into external authentication services such as LDAP, RADIUS, PAM and Kerberos. This database schema also allows for the management of users. Database control can be provided only giving users the access needed. PostgreSQL also supports the ANSI SQL standard which makes it easier for it to be integrated to store data from a resource such as OSQuery.

The use of PostgreSQL in application development is largely due to the rich tool sets on offer in order to create applications. These include the ability to support various server programming languages which means larger support for developers. It also provides APIs to access the DBMS from applications that are written in any language making use of the standard ODBC and JDBC interfaces. PostgreSQL also offers a wide selection of database objects. It has a built-in flexible full-text search system that supports a large set of languages. The database further supports semi-structured data (documents) which is similar to NoSQL databases. This allows for the wrapping of foreign data which permits the data to be ingested from external data sources. As previously mentioned, PostgreSQL

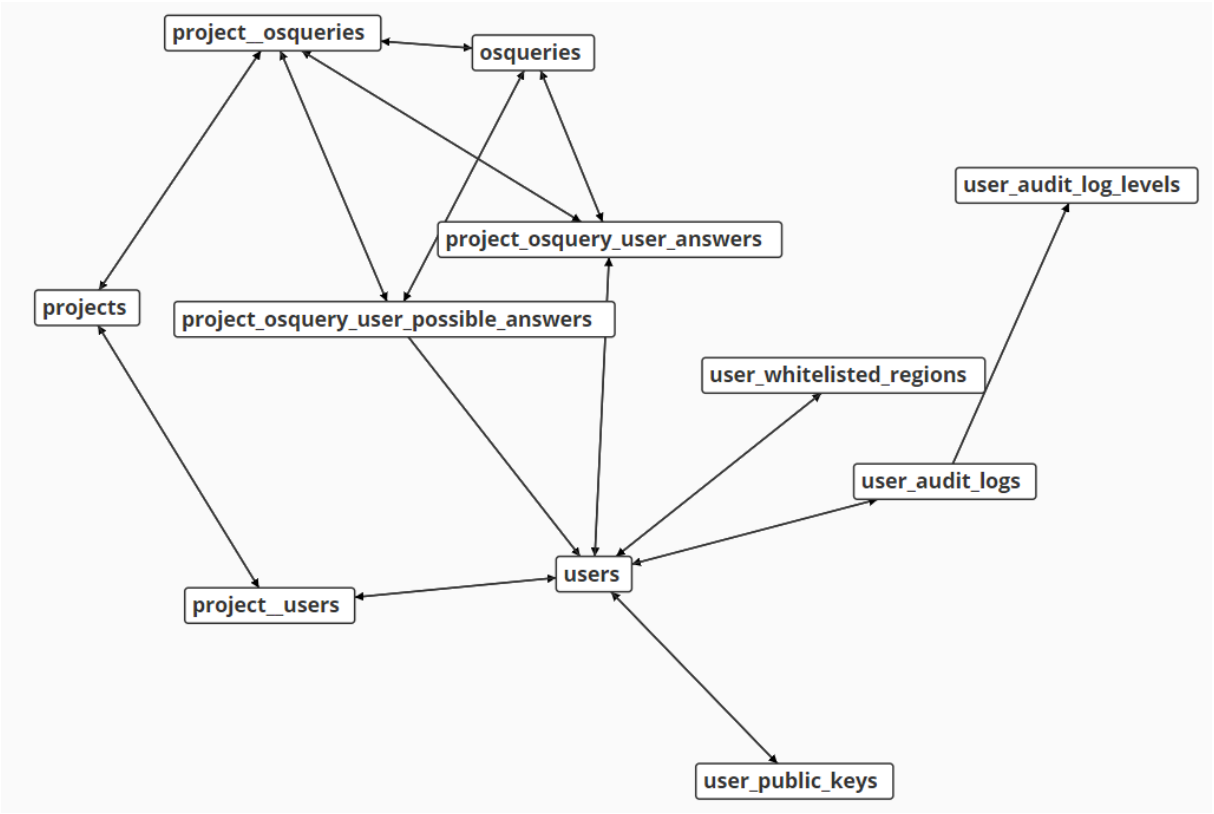


Figure 3.3: Code Corral - Database Overview (full ERD in Appendix B.5)

is a relational database so the main purpose is to aid in the rapid storage and retrieval of data.

For Code Corral a simple, relational database was required on the server side to store information about users (developers), projects (code repositories) and the queries (specifically OSQuery queries). Figure 3.3 illustrates that users and projects form the central ideas in the data structure but that there are other components as well related to the deployment of Code Corral as well as the ability for Code Corral to successful update itself.

For example the current, valid software and database schema versions are stored in the database and queried by the client regularly to ensure that the client auto-updates both when needed and that it cannot end up locked out of the system. Not having access to Code Corral could result in a developer being unable to work — a consequence to be avoided at all costs.

For users and projects the information stored in the database includes:

- For users (see figure 3.4):

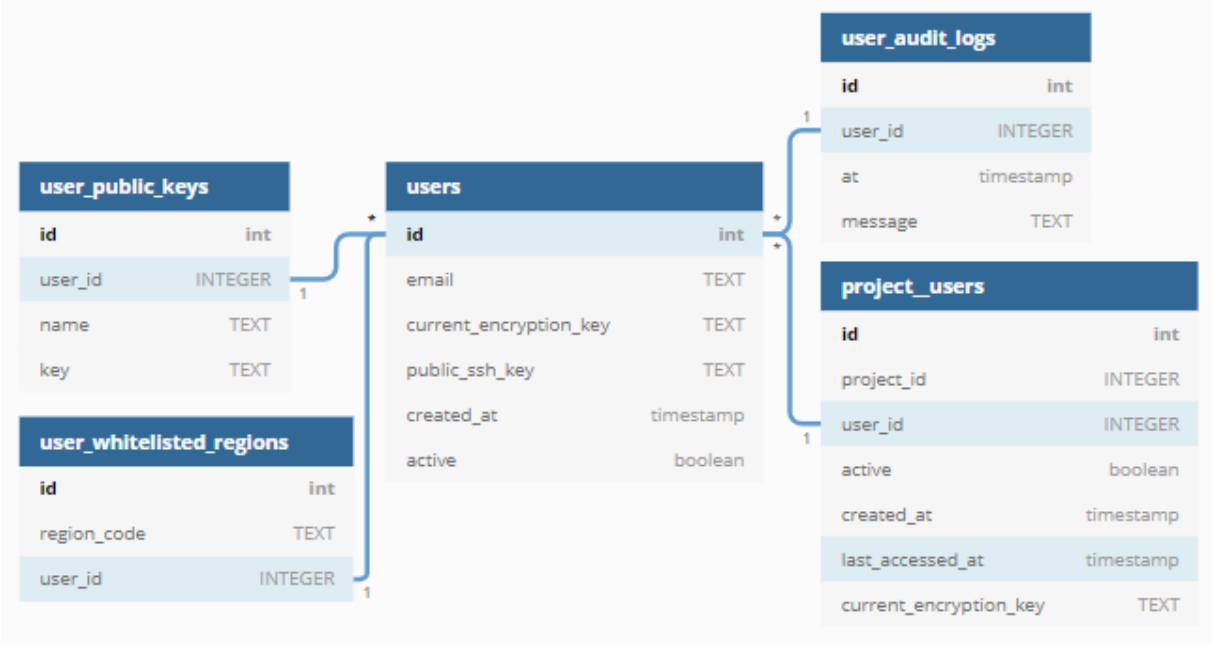


Figure 3.4: Code Corral - Users Relationships

  - SSH public keys for the user.
  - Accessible projects.
  - Permissible regions from which to access projects.
  - Previous answers to the queries for accessible projects.
  - Audit logs concerning access.

- For projects (see figure 3.5):
  - The queries required for the project.
  - The valid responses to the queries for this project.
  - The users granted access to the project.
  - The current encryption key for a given user for that project[5].

The data structure allows an organisation to create OSQuery queries once and then associate those queries with multiple projects. It is reasonable to assume that a user would want the particular query which confirms which OS is being run. Further, available details regarding the patch-level of that OS should be accessible to multiple projects — possibly even all the projects the organisation chooses to protect using Code Corral.

[5] This encryption key is a Globally Unique Identifier (GUID), randomly generated, and is therefore not encrypted in the database.

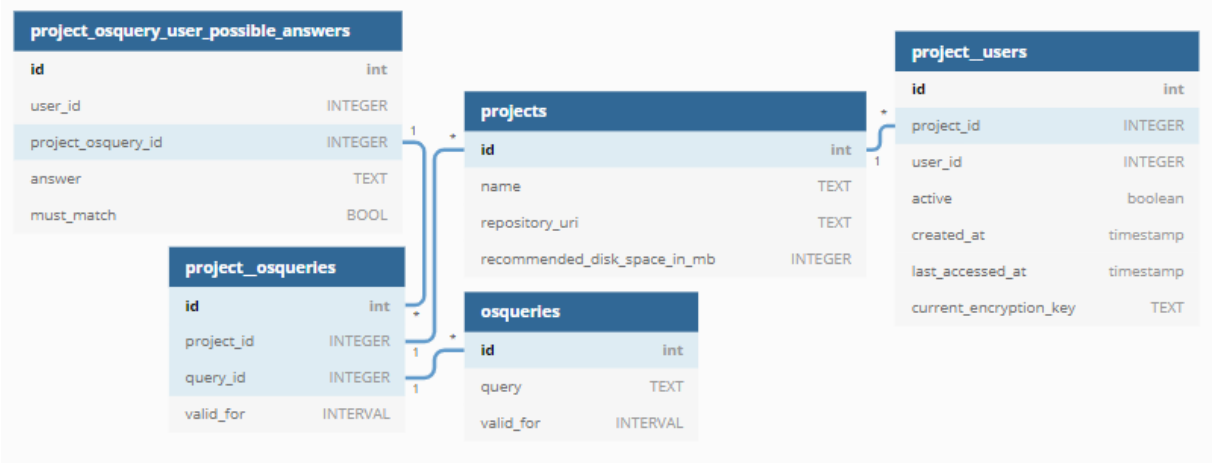


Figure 3.5: Code Corral - Projects Relationships

It is equally reasonable that one would require the ability to execute the same query for multiple projects but that different responses for different projects would be acceptable. For example, one might require the patch-level to be n for **Project X** but for **Project Y** it may be permissible to allow a patch-level of $n - 1$. The data structure allows for this, as can be seen in figure 3.5. The full entity relationship diagram (ERD) for Code Corral can be found in Appendix B.5.

Sinatra

Sinatra is a lightweight web framework that is written in the Ruby programming language. Sinatra is built on Rack which is effectively a bare bones version of Rails. Rack's functionality is similar to that of Rails, with the exception that it leaves a number of things for the developer to do. An example can be seen in the fact that Rack does not have the ability to create the file-structure for the developer. While the convenience of this function is sacrificed, the benefit is that since some tasks are performed manually, the result is a light weight server which contains only the necessary components.

Sinatra is a Domain Specific Language (DSL) which is used by Ruby to build web applications. The Sinatra framework was developed by Blake Mizerany in 2007 to allow people to create these applications with the least effort possible. The developer merely has to manually set up the routes and connect them so that the application knows how to connect to the database and which files to show in the browser. Sinatra is used by many top-tier companies such as Apple, BBC, GitHub and LinkedIn.

Making the choice to select Sinatra for Code Corral was easy once the choice of language

was set, for all of the considerations provided above. The server component of Code Corral is a web-service built as a Ruby app using a basic Model-View-Controller (MVC) pattern.

The MVC layout is apparent in the folder structure of Code Corral - as seen in figure 3.6. The **models** defined in the server-side code include definitions for queries, projects, and users – together with the relationships between those. They mirror the tables defined in section 3.1.2 and are the code representation of the stored data and can be reviewed in Appendix B.7.

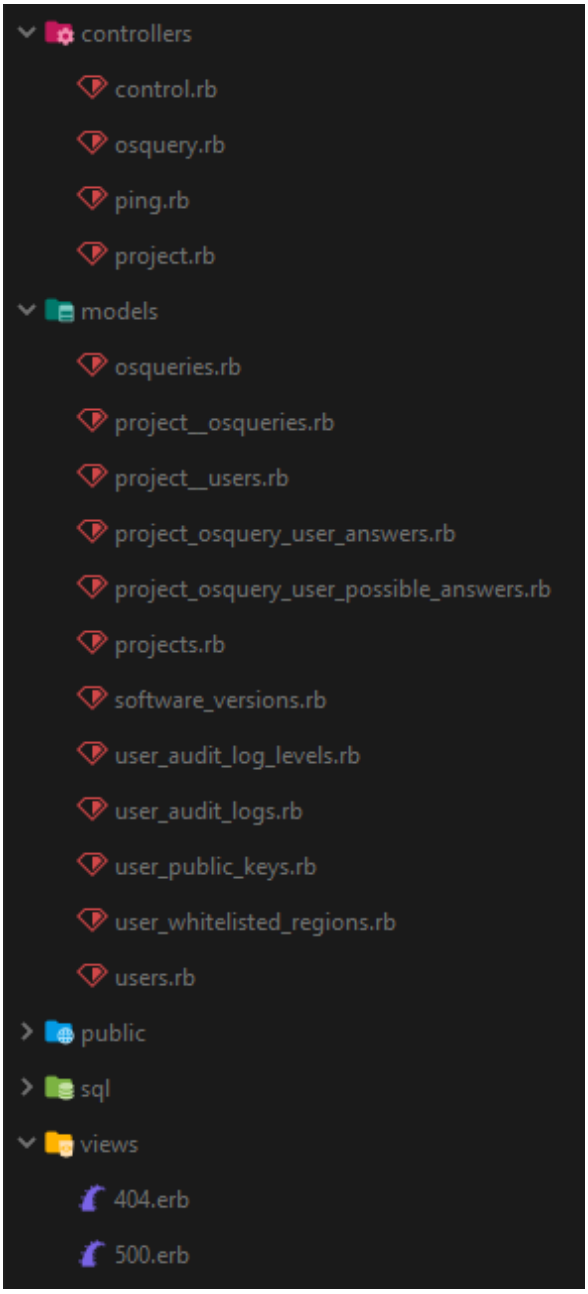


Figure 3.6: Code Corral - Files and Folders

There are only two **views** defined in Code Corral namely, one for *Not Found* and one for *Error*. If it did not do so the server would return commands to the `bash` client. The client uses those commands as parameters, which it then uses to continue interaction with the server. The two views (404, and 500) are included in Appendix B.7 for the sake of completion.

The server-side logic is defined in the *controllers* (fully included in Appendix B.8) albeit there are a few functions and features worth describing in more detail here. The three features the researcher would like to highlight are:

- `verified_region`;
- `verify_software_version`; and
- `project/open`.

verified_regions

This feature is visible in the except from

`control.rb` in listing 3.4 on lines 21 – 42.
The feature allows Code Corral administrators to specify regions of the world that requests to Code Corral should come from, for a given user. If a request comes from somewhere unexpected then it is rejected. This allows some basic geo-location information to be parsed by Code Corral and opens the solution up to more controlled environments. For example, Code Corral could be further locked down to allowing access only from predetermined IP addresses.

Listing 3.4: Code Corral / control.rb / excerpt

```
1  before %r|/api/1\.1(?:/.*)?| do
2    user_email_address = request.env['HTTP_X_CODE_CORRAL_EMAIL_ADDRESS']
3    halt( 400, "no email" ) unless user_email_address
4
5    # Yes we could include the client in the tooling area below, but we want
       to
6    # expand tooling into a data driven mechanic eventually rather than hard
7    # coded as it currently is.
8    client_version_hash = request.env["HTTP_X_CODE_CORRAL_CLIENT"]
9    halt( 400, "no client hash" ) unless client_version_hash
10   verify_software_version client_version_hash, "client"
11
12   %w|veracrypt osquery|.each do |tool_name|
13     software_hash = request.env["HTTP_X_CODE_CORRAL_TOOL_#{tool_name.upcase
       }"]
14     halt( 400, "no #{tool_name} hash" ) unless software_hash
15     verify_software_version software_hash, tool_name
16   end
17
18   auth = request.env['HTTP_X_CODE_CORRAL_AUTH']
19   halt( 400, "no auth" ) unless auth
20
21   code_corral_signature = Base64.decode64( auth )
22   halt( 400, "empty auth" ) unless code_corral_signature
23
24   @user = User.by_email( user_email_address )
25   halt( 400, "auth denial" ) unless @user and @user.active?
26
27   location = Geocoder.search( request.ip ).first
28   halt( 400, "unable to detect location from #{request.ip}" ) unless
       location
29
30   descriptive_location = "#{location.city} #{location.province} #{location.
```

```
    country}".strip
31  region_code = "#{location.country_code}_#{location.province_code}_#{
    location.postal_code}".upcase
32
33  @verified_region = nil
34  @user.whitelisted_regions.each do |whitelisted_region|
35    @verified_region ||= whitelisted_region if %r|#{whitelisted_region.
    region_code}| =~ region_code
36  end
37
38  unless @verified_region
39    puts "WARNING: region_code #{region_code} not allowed for #{@user.email
    }"
40    @user.audit_log :warning, "Access DENIED from #{descriptive_location},
    which could not matched '#{region_code}'"
41    halt( 400, "geo failure" )
42  end
43
44  @user.audit_log :info, "Accessing from #{descriptive_location}, which
    matched '#{region_code}' due to '#{@verified_region.region_code}'"
45
46  @verified_public_key = nil
47  @user.public_keys.each do |user_public_key|
48    public_key = OpenSSL::PKey::RSA.new( user_public_key.key )
49    unless public_key.public?
50      STDERR.puts "WARNING user public key is not public: #{user_public_key
    }"
51      next
52    end
53
54    begin
55      @verified_public_key = user_public_key if "#{@user.email}|#{
    client_version_hash}" == public_key.public_decrypt(code_corral_signature
    )
56    rescue OpenSSL::PKey::RSAError
57      next
58    end
59  end
60
61  unless @verified_public_key
62    @user.audit_log :warning, "No public key matched!"
63    halt( 400, "auth failure" )
64  end
65
```

```
66    @user.audit_log :info, "Verified public key: #{@verified_public_key}"
67  end
```

verify_software_version

In `control.rb` there is a function called `verify_software_version` the role of which is to ensure that the Code Corral client (and dependencies) are up-to-date and as expected, as can be seen in listing 3.5.

This function is called before attempting to authenticate a user or verify access to a repository as can be seen in lines 8 – 16 in listing 3.4 and checks initially to see if the hash for the client has changed and then to see if the hashes for any of the tools (currently `osqueryi` or `veracrypt`) have changed.

Listing 3.5: Code Corral / verify_software_version.rb

```
1  def verify_software_version( hash, return_enumerate )
2    software_version = SoftwareVersion.where( hash: hash ).first
3    update_required = false
4    if software_version
5      unless software_version.active?
6        update_required = true
7      end
8    else
9      update_required = true
10   end
11   if update_required
12     if return_enumerate == "client" # Client disabled or not found
13       halt( 400, "bad client" )
14     end
15   else
16     latest_software_version = SoftwareVersion.latest_by_name(
       software_version.name )
17     update_required = true unless latest_software_version ==
       software_version
18   end
19   if update_required
20     halt 200, "update|#{return_enumerate}"
21   end
22 end
```

If anything has changed, the client is either instructed to execute the upgrade scripts or it is simply denied access.

project/open

On cloning local repository the Code Corral server generates a unique password for the volume. That same process is performed in the `project/open` functionality in `project.rb` in listing 3.6. The volume is then opened using the old password and the password is changed to the new one. This means that when the volume in encrypted again the password used is not one that was previously know to the client.

Listing 3.6: Code Corral / project.rb / open

```
post %r|#{API_PREFIX}/project/open/([^/]+)| do |project_name|
  if @user_project.current_encryption_key == nil
    @user.audit_log :warning, "Attempted to open an uncloned project #{
    @project.name}"
    halt( 400, 'project not yet cloned' )
  end
  old_encryption_key = @user_project.current_encryption_key
  new_encryption_key = $uuid.generate
  @user_project.update( current_encryption_key: new_encryption_key )
  @user.audit_log :info, "Opened project #{@project.name}"
  "project|open|#{@project.name}|#{old_encryption_key}|#{new_encryption_key
    }"
end
```

3.2 Deployment

As mentioned in section 3.1, Code Corral is designed to be as cross platform as possible. To this end, it should run on the three major development environments i.e. Windows, macOS and Linux. For the experiment however testing was limited to Linux and Linux under Docker, as the modifications to scripting for macOS has been left as future work. Further, the many different ways in which Bash can be installed on Windows makes a solution for that platform complicated. Modern Windows deployments have the WSL as mentioned in section 3.1.1 and that when WSL 2.0 is installed then deployment there is identical to Linux (as it is in fact Linux running).

3.2.1 Client Deployment

The docker image utilised for workstation deployments only required `bash`, `curl`, `veracrypt`, and `osqueryi`. For the Linux deployment `bash` was assumed and the rest of the required

binaries are installed as part of the client installation, either by fetching them from the Code Corral server's public binary folder or by installing them through `apt-get`.

Instructions for deploying Code Corral to a machine are simple and included in the source repository as a Markdown file, `README.md` (see listing 3.7).

Listing 3.7: Code Corral / README.md

```
Client Installation
===================

```
wget -qO- https://code-corral.herokuapp.com/client/bash/code-corral | bash
```

Generating Public Key (OpenSSL format) for Code Corral Server
===

```
ssh-keygen -e -m pem -f ~/.ssh/id_rsa | openssl rsa -RSAPublicKey_in -
    pubout
```
```

The second half of the README.md in listing 3.7 covers creating a public key in the format expected by the Code Corral server.

3.2.2 Server Deployment

The server was deployed on Heroku[6], which is a Platform as a Service (PaaS) solution offering a free tier. It met the requirements the researcher had for the test deployment. Test deployment requirements included:

- **Heroku Run-time** for running the Ruby / Sinatra application;
- **Heroku PostgreSQL** to provide a managed database; and
- **GitHub Integration** and **Continuous Delivery** to provide this project with the ability to merge fixes and have the server deploy those updates immediately.

[6] https://www.heroku.com/

The deployment and configuration of the Sinatra application on Heroku is initiated as part of the integration with GitHub, triggered by a push to the master branch for a given GitHub repository, and uses the `app.json` file defined in listing 3.8 to configure and set up the application.

Listing 3.8: Code Corral / app.json

```
{
  "name": "code-corral-server",
  "scripts": {
    "postdeploy": "bundle exec rake db:schema"
  },
  "env": {
    "DATABASE_URL": {
      "required": true
    },
    "LANG": {
      "required": true
    },
    "RACK_ENV": {
      "required": true
    }
  },
  "formation": {},
  "addons": [
    "heroku-postgresql"
  ],
  "buildpacks": [
    {
      "url": "heroku/ruby"
    }
  ]
}
```

3.3 Summary

Code Corral provides a solution that leverages several open source projects (`osqueryi` and `veracrypt`) and wraps them in a simple `bash` script. Veracrypt provides the ability to encrypt the Git repository locally. This allows an organisation using Code Corral to revoke access to the locally stored copy, simply by not providing the system with the decryption keys. This allows for the fulfilment of the *Availability* for the source code remotely while

maintaining *Confidentiality* as required by the problem statement. OSQuery allows the tool to be tweaked to allow greater requirements to be met before access to the source code is granted, meaning that the level of *Confidentiality* can be adjusted.

As a PoC the solution is fairly robust, providing some self-healing capability and the ability to update the rules and to revoke access. There is much more that can be done and we will address some of this in chapter 6. Figure 3.2 gives an idea as to how the process of installing the Code Corral client and checking out the first repository would work.

Chapter 4

Testing

Testing is defined as a process in which an evaluation is carried out to determine whether a specific system meets the specified requirements or not. This process mainly encompasses the validation and verification during which it is determined if the developed system meets the requirements which were defined by the product owner in advance. When software is tested it is in an attempt to uncover bugs (errors), or components missing from the requirements of the developed system or software. The development of computer science has seen that it is not so much only the technical perspective of systems but the social consequence or acceptance of the proposed systems (Wienclaw, 2019) that matters most. The researcher is proposing that both aspects need to be explored when testing the proposed new solution, Code Corral.

The system has to be tested in terms of both its functionality and the technical elements which which make it up. Further, the all important aspect of its usability should also be piloted. That is to say that the social acceptability of the software also needs to be trialed. The proposed method for achieving this is the use of surveys. The surveys are to be sent out to developers to determine how the software is accepted by them.

This acceptance testing should be catered for with an enhanced and efficient methodology of testing. There are broadly two types of test types which are:

1. Functional testing (covered in section 4.1); and
2. Non-Functional testing (covered in section 4.2).

These will be explored to determine which approach would be best suited for testing the Code Corral software whilst taking certain constraints into consideration i.e. limitations on access to the development teams needed to implement the code; and time constraints in the execution thereof.

The intention is to leverage existing tools to the fullest extent. Essentially, avoiding unnecessary changes to the manner in which developers work. As an example, developers are accustomed to managing local copies of source code and using Git to push their updates to the central repository. The proposed system should allow that to continue, with as little modification to the process as possible, while still providing some level of additional protection for the source code. It should noticeably increase the *Confidentiality* afforded companies by securing the source code that developers have access to. Simultaneously, it should limit the perceived decrease in the *Availability* of the source code to developers.

The technical testing laid out in the sections below follows a slightly modified Gherkin syntax made popular by the Cucumber[1] project. Cucumber is an open source testing framework that draws heavily from the Agile Alliance's views on user acceptance testing[2]. The syntax is based on the use of keywords in the test description that make it easy to follow. The example given in the Gherkin documentation[3] looks like this:

```
Example: "Multiple Givens"
  Given one thing
  And another thing
  And yet another thing
    When I open my eyes
      Then I should see something
      And something else
        But I shouldn't see yet another thing
```

The slight modification to the Gherkin syntax relevant to this endeavour is to include the **AS** keyword. This is used to relate the tests to the role from which this research is viewing it and high-lights the importance of said view-point.

Use cases are tests described from the perspective of imagined users (actors or roles) interacting with the product in a *As* / *Given* / *When* / *Then* manner. That is to say, the test definition is written as (for example):

[1] https://cucumber.io/
[2] https://www.agilealliance.org/glossary/gwt/
[3] https://cucumber.io/docs/gherkin/reference/

```
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "Example scenario"
  Given that I have access to a repository
    When I clone it through Code Corral
      Then it clones successfully
      And I'm able to use it
```

Positive and negative testing (Czerwonka, 2005) is when tests prove that the software works as expected (positive testing), but also (and perhaps more importantly) that the software is able to work in situations of invalid inputs or user behaviour that could not be predicted. Any endeavour wanting to show the good working of software would need to include both types of tests. Referring to the previous example, a second use case that might be of interest might be phrased as:

```
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "Negative example scenario"
  Given that I do NOT have access to a repository
    When I clone it through Code Corral
      Then it should give an error
      But it should NOT clone
```

For the tests detailed below the actors are: the developer using the Code Corral client, and the Code Corral administrator who is setting Code Corral up and managing it.

4.1 Functional Testing

The functional testing of software is analysed in four components. These components are:

1. Unit Testing (section 4.1.1);
2. Integration Testing (section 4.1.2);
3. System Testing (section 4.1.3); and
4. User Acceptance Testing (section 4.1.4)

It is good practice to execute various combinations of these tests. Below, each of these components will be further explored.

4.1.1 Unit Testing

Unit testing methodology aims to test individual software modules, functions or components that make up the application or system. This testing methodology is embraced by development processes like Agile, Scrum and eXtreme Programming (XP) and is usually written prior to the creation of a module when test driven development (TDD) is employed. Each module, component or function is then tested against a specific unit test to determine whether the module is functioning as it should. The unit testing of software is done during the development of the application and ensures that each component does what is expected of it. The main objective is to test isolated portions of the code and verify their correctness.

The benefits of doing unit testing of code (Runeson, 2006) is that each test is able to test the functionality of each unit to determine if they each function independently the way they should. It allows the developer to have an understanding of which aspects of the functionality currently works. This allows for a baseline for the purpose of regression testing at a later stage, once his updates have been pushed, to determine if that functionality remains the same. This is a modular method of testing and means that parts can be tested independently from each other and not with any reliance on the whole development needing to be complete. The disadvantages to this method of testing is that each portion is tested in isolation. Further, it will not identify integration errors or broader system level errors.

Unit tests were not created for Code Corral owing to time constraints. In addition, the initial development project only had a single developer working on the code at the given time. The development was brought about as a PoC — with the understanding that the code may end up being thrown away completely and rebuilt from scratch if it was found to be a good platform. Regression testing is more important in larger teams with multiple commits modifying the codes base all the the time, but for future work definitely unit tests.

4.1.2 Integration Testing

On completion of the unit testing, the next test batch used to test the source code is integration testing. The modular components are integrated to create the various modules that encompasses the components of the newly designed platform. These are then group tested to determine whether, as a collective, they integrate and act in a manner which is expected. These tests will often be framed in terms of user scenarios, which can be as simplistic as: "can the software log an authorised user in". It allows the tester to determine if the software can do basic functions which it proposes it can. There are various approaches to integration testing of software where the collective is tested. These are:

- **The Big Bang approach** combines all the units simultaneously. This approach is taken when the testing team would receive the bundled software in its entirety. Consequently, no individual modules are tested but rather, the software is tested as a whole.
- **The Top Down approach** tests the top-level units first. This is generally used in cases where a top-down development approach is taken. The testing is performed incrementally and testing stubs are needed to simulate the lower-level units.
- **The Bottom-Up approach** is taken when code has been developed from the bottom up. In this instance, it is important to have test drivers that simulate higher level units which are needed and may not be available at this phase of development.
- **The sandwich/hybrid methodology** of integration is a methodology that includes a combination of Top-Down and Bottom-Up approaches.

It is important to test how effectively various components of the software integrates with each other and functions in the way in which they are expected to function. This can be achieved by ensuring that the correct versioning of each element is detailed in your user stories. The interaction between each element should be taken into account.

Access Control

Code Corral provides access control as a complete service, but for this integration test the reasearcher reviewed four use cases. As discussed in section 4.1.2, the research discusses tests which cover a number of different roles' perspectives. The use cases follow.

The test that follows is used to set up the rest of the tests in this batch. As an administrator, the tester will grant a public key access to a repository and then monitor the server-side logging to ensure that the request made by that user is properly identified as valid and that the server responds by providing access to clone the repository.

```
As an administrator of Code Corral
-----------------------------------------------------------
Scenario: "Authorised user clones a repository"
  Given I have granted a user access to a repository
    When they attempt to clone the repository
      Then they are granted access to clone it
```

This next test starts with the set-up described above and follows the client-side of the interaction between the developer's machine and the cloning request.

```
As a developer using Code Corral
-----------------------------------------------------------
Scenario: "Authorised user clones a repository"
  Given I am granted access to a repository
    When I clone the repository
      Then Code Corral client starts the clone function
```

This test below assumes that the previous two tests are successful and has the test execute from that point. This test result is reviewed from the server side and verifies whether when access to open the repository was granted, is the secret for the encrypted volume returned to the client.

```
As a developer using Code Corral
-----------------------------------------------------------
Scenario: "Authorised opening a repository"
  Given I have cloned a repository and have access
    When I open the repository
      Then access should be granted
```

The final test for this section is the negative use case of the previous test. When the same request is made to open a repository after access was revoked is the client told that access is no longer possible?

```
As an administrator of Code Corral
```

```
    ---------------------------------------------------------
    Scenario: "I revoke access to a user and they attempt to open"
      Given access was possible to repository
      But it has now been revoked
        When a user attempts to open the repository
          Then the user should be denied access
```

Data Collection

A large part of what the client does is execute OSQuery queries through `osqueryi` at the request of the server and then send the responses back to the server for evaluation. To incorporate the multiple perspectives angle talked about in section 4.1.2 both the client-side of that equation as well as the server-side is going to be looked at.

```
    As a developer using Code Corral
    ---------------------------------------------------------
    Scenario: "OSQuery of a value succeeds"
      Given that the client is correctly installed
      And I've asked for access to a repository
        When I received a request from to look up a value using OSQuery
          Then OSQuery should return the value requested
```

This tests the integration with OSQuery on the client-side and ensures that the queries are parsed correctly by the `osqueryi` client.

```
    As an administrator of Code Corral
    ---------------------------------------------------------
    Scenario: "OSQuery of a value succeeds"
      Given a request triggered an OSQuery lookup
        When the data is returned
        And it matches the required value
          Then the server should return SUCCESS
```

This is the server side of the previous test and allows the tester to verify that the results returned are used to correctly grant access to the user

```
    As an administrator of Code Corral
    ---------------------------------------------------------
```

```
3     Scenario: "OSQuery of a value fails"
4       Given a request triggered an OSQuery lookup
5         When the data is returned
6         And it does NOT match the required value
7           Then the server should return FAILURE.''
```

This includes the negative test requirement that was set in section 5.1 by allowing the tests to ensure that the server does differentiate between successful query responses and failures.

Volume Mount / Dismount

The other integration that Code Corral has is with the VeraCrypt client (`veracrypt`). This test is simply to ensure that those commands are executed correctly.

```
1     As a developer using Code Corral
2     ---------------------------------------------------
3     Scenario: "Cloning a repository succeeds"
4       Given a repository is not yet cloned locally
5         When a clone request is sent
6         And access is allowed
7         And cloning permitted
8           Then VeraCrypt should create and mount a volume for this purpose
```

This tests not only the ability of VeraCrypt to mount volumes on the client machine but it also checks to ensure that the logic built in concerning volume creation and the size of that volume is correct.

```
1     As a developer using Code Corral
2     ---------------------------------------------------
3     Scenario: "Volume mount succeeds"
4       Given a locally cloned repository
5         When an open command is requested, authorised and executed
6           Then the volume should successfully mount
```

This tests the ability of the client to recognise that the volume exists already and to mount it successfully.

```
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "Volume dismount succeeds"
  Given a locally cloned repository
    When the close command is requested
    And it is executed
      Then the volume should successfully dismounts
```

This tests the ability of the client to dismount volumes previously mounted.

4.1.3 System Testing

Once the integration test is completed, it should be determined whether the software functions as a comprehensive unit. The purpose of this test is to evaluate whether the system is compliant with the specified requirements. This process is often referred to as the Black Box testing method (Khan, Khan *et al.*, 2012). This is also referred to as a behavioral test. As the name suggests, it regards the behavior of software. The elements which are tested with this method are:

- Incorrect or missing functionality
- Interface errors
- Errors in the data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

This method of testing is done from the user's perspective, i.e. from the viewpoint of the user to determine if they can use it the purpose it was designed for. It can be performed by the developer or an independent non technical person. No programming knowledge is needed to test whether the software behaves correctly. The disadvantage of system testing is that it is not always possible to test all the inputs and some portion could be left untested. However, if a developer is aware of these constraints, and has implemented decent bug reporting elements then, when uncovered, these can be corrected with updates. As long as a detailed change-log is kept, this can be monitored.

System testing typically covers more than one user story and more than one piece of functionality is tested. A batch of user stories that are related in some way are often referred to as an epic[4] and so most often system testing will cover a whole (or possibly more than just one) epic.

Deploy a Server

This epic takes an administrator through the steps of deploying a new server as well as the configuration thereof. Typically, this instance would be following the steps documented as part of development, but for this project those steps were manually performed and the test itself used to write the deployment guide.

Deploying a Code Corral server has three required steps and then a optional one. The required steps to complete are:

1. Set up the RDMS — for Code Corral this is PostgreSQL;
2. Set up the web-server — for Code Corral this is Sinatra;
3. Populate the database with projects, users, and queries

The optional step is to configure the Code Corral server environment to be managed via a continuous integration / continuous deployment (CICD) pipeline. If the test were to be described here it would look like this:

```
1   Feature: "Installing a Code Corral server"
2   As a administrator of Code Corral
3   ---------------------------------------------------
4   Scenario: "1. Set up the DBMS"
5     Given a Linux server with PostgreSQL
6     (or a cloud provider with a managed RDMS solution)
7       When all the scripts in ./sql are executed
8         Then the database schema should match the models in ./models
9   ---------------------------------------------------
10  Scenario: "2. Set up the web-server"
11    Given a Linux server with Ruby
12    (or a cloud provider with Ruby)
```

[4] https://www.atlassian.com/agile/project-management/epics

```
13          When dependencies are installed
14          And database migrations run
15          And deployment complete
16            Then the web-server should return "no email" to a GET request:
17            from "https://code-corral.herokuapp.com/api/1.1"
```

Configure a Repository

When a Code Corral server has been installed it needs to be populated with users, projects and rules about how those projects are able to be accessed.

```
1     Feature: "Installing a Code Corral server"
2     As a administrator of Code Corral
3     ---------------------------------------------------
4     Scenario: "1. Add users"
5       Given an installed Code Corral server
6         When a database client is connected to the DB
7         And user details are added
8           Then the database should store those values
9     ---------------------------------------------------
10    Scenario: "2. Add projects"
11      Given an installed Code Corral server
12        When a database client is connected to the DB
13        And project details are added
14          Then the database should store those values
15    ---------------------------------------------------
16    Scenario: "3. Add rules (queries)"
17      Given an installed Code Corral server
18        When a database client is connected to the DB
19        And query details are added
20          Then the database should store those values
```

This test primarily sets the system up for full testing. Once an administrative user interface (admin UI) or an advanced programming interface (API) exists, then these tests will be required for testing those components of the solution.

Install Client

This is a fairly simple test to execute but it ties together multiple parts of the system in order to work. The client install should be as simple as running a single command line

as per the README.md file but it should result in not only Code Corral being installed but also VeraCrypt and Osquey as well.

```
Feature: "Installing the Code Corral client"
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "1. Fetch to Code Corral client"
  Given an installed Code Corral server
    When the install command is executed
      Then the Code Corral client should execute
-------------------------------------------------------------
Scenario: "2. Install Curl"
  Given curl is not installed
    When Code Corral client installs
      Then curl should be installed
-------------------------------------------------------------
Scenario: "3. Install VeraCrypt"
  Given veracrypt is not installed
    When Code Corral client installs
      Then veracrypt should be installed
-------------------------------------------------------------
Scenario: "2. Install OSQuery"
  Given osqueryi is not installed
    When Code Corral client installs
      Then osqueryi should be installed
```

Checkout a Repository

Now that a configured server exists and a working client installation is in place, the next tests look to see if a repository can be checked-out for use. This will verify that the local tools are all installed correctly and that the configuration done on the server is complete.

```
Feature: "Using the Code Corral client"
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "Clone a repository"
  Given a configured Code Corral server
  And an install Code Corral client
    When a clone command is executed for a repository
    And the developer has access to that repository
```

```
9           And meets the requirements for access
10            Then a volume should be created
11            And the volume should be mounted
12            And the repository should be cloned into the volume
13      ------------------------------------------------------------
```

4.1.4 User Acceptance Testing

During the final software development phase, user acceptance of the software is sought by inviting users to test the software. Application testing or end-user testing as referred to above, is a phase of software development in which the software is tested for "real-world" use for a specific audience. The test affords users the opportunity to interact with the software before the official release in order to determine whether any features have been overlooked or to identify unknown bugs. There are various methods which can be used during this testing phase. Some of these methods are:

- **Beta Testing** (β) which is where software is given to a group of end users, who will be responsible for using the software as it was intended to be used. The users then need to provide feedback to the developers to make changes or improvements. An earlier phase of beta testing is called alpha testing (α) and this usually maps to system testing as described in section 4.1.3.
- **Operational Acceptance Testing** (OAT) which places the focus on proper workflow for the software rather than real-world testing.
- **Contract Acceptance Testing** in which the software is tested against certain contractual or industry standards.
- **Regulation Acceptance Testing** which is the testing of the software in how effectively it meets certain regulatory requirements.
- **Functional Testing** is often the preferred method of testing for software development, due to it being a more effective; measuring how the software is intended to be used rather than looking at the parts. The reason for its effectiveness is the focus on the requirements it needs to satisfy. Does its functionality work the way it is intended to work? The main goal is thus testing whether the software functions the way it was intended.

The researcher had chosen to perform some specific functional testing and to gather the results of those tests in the real world through a series of surveys.

For the experiment two groups of interesting individuals were selected: developers that fulfilled the criteria below; and individuals that own the code those developers work on.

The developers involved had to be:

- familiar with Git as a source code repository;
- comfortable with using new tools; and
- happy to provide feedback on security tools.

The code owners had to have an understanding of the importance of securing code and be open to having their projects protected by the proposed solution. They needed to be:

- security aware;
- with an opinion on security matters; and
- invested in improving security for their code.

A number of teams within the same organisation were selected in order to simplify data collection.

Research Methodology

Qualitative research involves using different methods which range from participant observation, interviews, action research, document review and grounded theory to investigate or explain phenomena of human perspective. In this instance, the term methodology is used to refer to the way that the research will be conducted (Wienclaw, 2019). This covers the process which will be undertaken when collecting data for the survey. The aim is to accurately describe and list the parameters which led to certain sets of data being collected and others not.

Qualitative research in terms of this research is undertaken to understand the effects that this proposed system would have on a small subset of individuals based on their experience

within the field of development and with various aspects of security. The purpose of choosing a small subset of higher quality individuals to measure the acceptability of Code Corral within the problem space was to gather accurate high-quality testing.

The survey was intended to run for the duration of the research, for participants that fit the criteria as set out in section 4.1.4. There were limitations to the acceptability testing method which was chosen in that the results were dependent on the developers completing the survey. This was also a limiting factor in that the researchers were relying on participants to want to complete the survey and submit it. An evaluation was be done at the end of the chapter to establish whether this goal has been satisfied.

Research Instruments

Three surveys were administered online using the Google Forms platform and a fourth was administered one-on-one for respondents that volunteered. The questions of these surveys have been transcribed on the following pages, as well as the kind of answer that the survey allowed. Each automated survey was prefaced by the following text, which the individual agreed to in order to fill out the survey:

> *You are invited to participate in a research study entitled "Securing Software Developer Using Developer Access Control" that forms part of a Master's thesis in Information Security. The aim of this research is to determine whether better Confidentiality of source code can be achieved while ensuring developers Availability to same. Your participation and cooperation is important so that the results of the research are accurately portrayed.*
>
> *Your identity and that of your institution will be treated with complete Confidentiality. The survey is completely anonymous. If you choose to discuss your results further, your identity will nevertheless be kept strictly confidential. The results of the survey will be aggregated and further made anonymous before inclusion in any publications.*
>
> *Your participation is completely voluntary and you may withdraw from the survey at any time without penalty. Your decision to participate or not entails no risk, penalty, or alteration of benefits. Furthermore, it is important that you are aware that this study has been approved by a Research Ethics Committee of Rhodes University, and informed consent has been obtained from your*

organisation. Filling out the survey will require about 15 minutes of your time to complete.

To continue filling out the survey, you must indicate your acceptance and understanding of the above text by checking the "I agree to participate under these conditions" checkbox below.

Ethics Considerations

There are numerous considerations in terms of ethics in relation to studies where surveys are used.

When collecting information from participants, consent must be explicit, informed, voluntary and re-negotiable. This means that by completing the survey participants agree to giving their consent however this consent can be withdrawn at any point.

Participants should always be given enough information to make an informed decision as to whether they are willing to participate in the data collection or not. Privacy is a further consideration which should be taken into account with regards as to how data is collected, what data is collected, how the data is stored and how it is reported on. The important consideration here is that the participants' anonymity is protected at all times. This means that where possible this research will make use of blind data so as to not be subjected to prejudice of any kind.

In addition to the general ethics guidelines Rhodes provides, the University of Sheffield[5] has guidelines that are well regarded covering research ethics associated with online survey designs. These include the following considerations which should be taken into account when creating an online survey:

1. A clear description should be given to the participant on what the study is about and any possible risks to the participant (if there are any).
2. A privacy statement should be included in your questions making certain that participants are fully aware of how the data collected will be kept and stored.
3. The participants should be informed of the tool used to gather the survey data in such as Mailchimp or Google Forms.

[5] https://www.sheffield.ac.uk/polopoly_fs/1.283716!/file/AGuideToResearchEthicsForOnlineSurveyDesign3.pdf

4. The participants should be offered the option of withdrawing from the survey at any time.

5. The participants should be made aware of the distinction between anonymity and *Confidentiality*.

6. The survey should be clear in the questions that it asks and the answers should be standardised.

In terms of use of a survey an Ethics approval process was completed and ethics clearance was awarded by the Rhodes University for the continuation of the survey. This can be seen in Appendix A.1.

Developer Survey - Baseline

General Information

1. **Of which team are you a member?** (Answer in the form of free-form text)

Git knowledge

2. **How usable do you find Git to be for source control?**
   (Answer as a 4-point Likert scale, going from "Not at all usable" to "Extremely usable"; with a free-form field for comments)

3. **Have you ever used other version control systems before Git?**
   (Answer as a choice between "Yes" and "No"; with a free-form field for comments)

4. **How long have you been using Git?**
   (Answer as a choice between "less than 1 year", "less than 3 years", "less than 5 years", "less than 10 years", and "more than 10 years")

5. **Have you ever had to perform any of the following tasks in Git? (Select all that apply)**
   (Answer as a multi-selection of "'init' a repository", "'push' changes to a remote", "Fix merge conflicts", "'rebase' a branch", "'cherry-pick' commits", "Manage access to a Git repository" and "Install a Git server")

6. **How would you rate your own competency with Git?**
   (Answer as a choice between options: "Junior", "Mid-level", "Senior", "Specialist"; with a free-form field for comments)

Effectiveness and Needs

7. **How effective do you think you are in your current role?**
   (Answer as a 4-point Likert scale, going from "Not at all effective" to "Extremely effective"; with a free-form field for comments)

8. **Are your needs met by the current tooling and processes?**
   (Answer as a 4-point Likert scale, going from "Not at all met" to "Fully met"; with a free-form field for comments)

Follow-up Questions

9. **Would you be willing to discuss your answers and related matters in a one-on-one setting?**
   (Answer as a choice between "Yes" and "No"; with a free-form field for comments)

Contact Details

10. **If you elected to allow us to contact you about your answers, please provide us with your contact details.**
    (Answer as 3 text fields, requesting "First name", "Last name", and "Email address")

Developer Survey - Post Testing

General Information

1. **Of which team are you a member?**
   (Answer in the form of free-form text)

Git knowledge

2. **How usable do you find Git to be for source control?**
   (Answer as a 4-point Likert scale, going from "Not at all usable" to "Extremely usable"; with a free-form field for comments)

3. **Have you ever used other version control systems before Git?**
   (Answer as a choice between "Yes" and "No"; with a free-form field for comments)

4. **How long have you been using Git?**
   (Answer as a choice between "less than 1 year", "less than 3 years", "less than 5 years", "less than 10 years", and "more than 10 years")

5. **Have you ever had to perform any of the following tasks in Git? (Select all that apply)**
   (Answer as a multi-selection of "'init' a repository", "'push' changes to origin", "Fix merge conflicts", "'rebase a branch", "'cherry pick' commits", "Manage access to a Git repository" and "Install a Git server")

6. **How would you rate your own competency with Git?**
   (Answer as a choice between options: "Junior", "Mid-level", "Senior", "Specialist"; with a free-form field for comments)

Effectiveness and Needs

7. **How effective do you think you are in your current role?**
   (Answer as a 4-point Likert scale, going from "Not at all effective" to "Extremely effective"; with a free-form field for comments)

8. **Are your needs met by the current tooling and processes?**
   (Answer as a 4-point Likert scale, going from "Not at all met" to "Fully met"; with a free-form field for comments)

Code Corral General

This section concerns the usage of the Code Corral software and the impression you have of it.

9. **Was the software installed for you or did you install it yourself?**
(Answer as a choice between "Installed it myself" and "Installed by someone else"; with a free-form field for comments)

10. **Did you find the software easy to use?**
(Answer as a 4-point Likert scale, going from "Dead easy" to "Far too complicated"; with a free-form field for comments)

11. **Did using the software impact your access to the source code?**
(Answer as a choice between "Yes" or "No"; with a free-form field for comments and a new question section "" being requested if "Yes" was selected)

Code Corral Impact

12. **Were you online and able to access the internet at the time?**
(Answer as a choice between "Yes" or "No"; with a free-form field for comments)

13. **Did you feel that Code Corral gave you a clear reason for blocking your access? Include the error message in the comment please.**
(Answer as a choice between "Yes" or "No"; with a required free-form field for comments)

14. **Did you feel that Code Corral was right in denying you access to the code?**
(Answer as a choice between "Yes" or "No"; with a free-form field for comments)

Code Corral Usefulness

15. **How would you describe Code Corral's usefulness?**
(Answer as a choice between options: "Not at all useful", "Somewhat useful", "Mostly useful", "Very useful"; with a free-form field for comments)

16. **How would you describe Code Corral's effect on your ability to work?**
(Answer as a choice between options: "Blocked me from working", "Slowed my ability to work down", "Didn't affect my ability to work", "Made working easier"; with a

free-form field for comments. “Blocked” and “Slowed” -> “Negative Impact”, “Easier” -> “Positive Impact”)

Code Corral Negative Impact

17. **As an estimated percentage of the planned sprint points, how much do you think that Code Corral impacted your ability to work?**
(Answer as a 4-point Likert scale, going from “Less than 10%” to “More than 90%”)

18. **What do you think could be done in Code Corral to lessen that impact?**
(Answer as a free-form field)

Code Corral Positive Impact

19. **Which feature of Code Corral made working easier for you?**
(Answer as a free-form field)

20. **What do you think could be done in Code Corral to make it more useful?**
(Answer as a free-form field)

Follow-up Questions

21. **Would you be willing to discuss your answers and related matters in a one-on-one setting?**
(Answer as a choice between “Yes” and “No”; with a free-form field for comments)

Contact Details

22. **If you elected to allow us to contact you about your answers, please provide us with your contact details.**
(Answer as 3 text fields, requesting “First name”, “Last name”, and “Email address”)

Developer Survey - Additional Questions

Git knowledge

1. How usable do you find Git to be for source control?
   (a) **Do you have another, preferred source control method?**
   (b) **Have you worked on projects with a large number of contributors before?**
      i. **If so: what tooling was used in that instance?**
2. Have you ever had to perform any of the following tasks in Git? (Select all that apply)
   (a) **Where / are you the go-to person for Git in your team?**
   (b) For the ones you have used:
      i. **How often do you feel you use them?**
      ii. **Do you refer to documentation about their usage when you use them?**
   (c) For the ones you haven't used:
      i. **Why not?**
3. How would you rate your own competency with Git?
   (a) **What would you consider to be very competent with Git?**
   (b) **How competent with Git do you feel you should be?**

Effectiveness and Needs

4. Are your needs met by the current tooling and processes?
   (a) **Are there security needs you see that are not currently being met with existing tools and processes?**
      i. **What are those?**
      ii. **How might they be met?**

Code Corral Impact

5. Did you feel that Code Corral gave you a clear reason for blocking your access? Include the error message in the comment please.
   (a) **What might have been more clear?**
   (b) **Would a more clear message have changed how you felt about being denied access?**
6. Did you feel that Code Corral was right in denying you access to the code?
   (a) **What reasons do you feel are valid for being denied access to the code?**

Code Corral Usefulness

7. How would you describe Code Corral's usefulness?
   (a) **What features would you like to see added to Code Corral?**

Management Survey

Security Experience

1. **What are your views on cyber security?**
   (Answer as a free-form field)

2. **Have you ever experienced what you would describe as a security incident or breach?**
   (Answer as a choice between "Yes" or "No" with Yes leading to "Security Breach Information" and "No" straight to "Security Stance")

Security Breach Information

3. **Was code lost as a result of the breach?**
   (Answer as a choice between "Yes", and "No"; with a free-form field for comments)

4. **Was there financial or damage to the company's reputation as a result of the breach?**
   (Answer as a drop down containing: "Financial loss", "Damage to reputation", "Neither", and "Both")

5. **Where you held responsible for the breach?**
   (Answer as a choice between "Yes", "No", and "Partially"; with a free-form field for comments)

Security Stance

Given the CIA triad of *Confidentiality*, *Integrity*, and *Availability* as described in more detail here: https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA how would you prioritise the three elements of security?

6. **Rank the following in order of importance** (table: 4.1[6])

[6] Arrange these three options in order of preference

Table 4.1: Ranking CIA by Priority

	First Priority	Second Priority	Third Priority
Confidentiality			
Integrity			
Availability			

Scenarios

Given the following scenarios please rate your confidence level in the statement below for each scenario.

7. **How confident are you that your source code is unlikely to be exposed?** (table:4.2[7])

Table 4.2: Confidence per scenario

	Not at all confident	Somewhat confident	Fairly confident	Extremely confident
Given: Code is unencrypted on the developer machine and access to the central repository is managed through ssh keys				
Given: Code is stored in an encrypted Code Corral volume and mounting that volume requires an ssh key check against the repository.				
Given: Code is stored in an encrypted Code Corral volume and mounting that volume requires a three-fold verification: 1.) an ssh key check against the repository; 2.) a confirmed location check against the allowed location list; and 3.) a check to ensure that the system is patched to the expected level.				

Follow-up Questions

8. **Would you be willing to discuss your answers and related matters in a one-on-one setting?**
(Answer as a choice between "Yes" and "No"; with a free-form field for comments)

[7] Section 4.1.4 gives rational for a four point Likert scale

Contact Details

9. **If you elected to allow us to contact you about your answers, please provide us with your contact details.**
   (Answer as 3 text fields, requesting "First name", "Last name", and "Email address")

Survey Data Collection Rationale

A number of different types of possible answer structures were selected for the survey. Each of these impact the values of the answers given in different ways:

- In the instance that a Likert scale is offered to respondents, it is always a scale of four, two positive and two negative in order to have measurable change with each answer. The most positive and most negative each move the dial two positions in the respective direction, while the other two each only one position.
- Binary questions are either the equivalent to a two point shift in the direction of positive or negative, or they serve to direct questions down alternative question paths.
- Trinary answers are like binary ones except that the third option is neutral and moves the dial not at all.
- Multiple choice answers are scalar and move the position of the dial only in one direction, ascending or descending in order of the possible answers.
- free-form answers are offered to allow respondents to provide context to their answers, but these answers do not impact the calculations done.

The questions are designed to collect data from a broad a spectrum of developers and interested third parties. In addition, the questions are curated to allow the answers of developers with experience working with development frameworks, the tooling around then, and knowledge of secure development practices to weigh more heavily than those with less, or none at all.

Git knowledge

Establishing a baseline for knowledge of Git and the related tooling is essential to understanding how the addition of Code Corral has impacted the developers' abilities to use their tool sets. Obtaining an understanding of the base knowledge of these tools for each participant is important to allow the surveyor to align, compare and contrast the answers for the rest of the survey. These questions are asked both in the baseline survey and in the post testing one. The reason for this is to account for knowledge of the tooling to have changed during the course of the testing.

How usable do you find Git to be for source control?

If a developer does not find Git to be a useful tool then it is likely they will find the Code Corral layer on top of Git to be a greater complication. The more usable they find Git to be as a solution the more they may see the value in Code Corral.

Have you ever used other version control systems before Git?

As with the previous question, this establishes the participant's knowledge of the possible options for managing version control available. If they have not used other systems then knowing the level of possible impact that Code Corral might have on the usability of version control will also be lessened.

How long have you been using Git?

An extended period using a system provides some indication of the level of comfort and competence someone might have with the system. It certainly establishes a timeline for features that Git offers with which the participant may be familiar.

Have you ever had to perform any of the following tasks in Git?

The list of tasks submitted cover a broad spectrum of actions within Git and establishes how intimately familiar the participant is with it as system. By increasing levels of knowledge regarding workings of the system, the options presented allow the surveyor to gauge the level of hands-on knowledge available.

How would you rate your own competency with Git?

This is a subjective question designed to establish the participant's level of confidence. High levels of confidence signifies a lesser likelihood of requiring assistance, and subsequently that the participant will be able to get a better view on how much the system impacted work.

Effectiveness and Needs

This section of the survey is provided by an organisation that uses it to determine the effectiveness of process changes within their development organisation. As such, the questions are intentionally aimed to elicit a reaction from the respondent that allows the surveyor to determine the effect this change has had on their ability to use the tools provided. Similar to a number of the other questions, a baseline was taken from respondents as part of the pretesting survey for comparative purposes.

How effective do you think you are in your current role?

Personal views on effectiveness form an integral part in establishing the developer's levels of job satisfaction. If a developer feels ineffective it will affect how satisfied they are with their work situation.

Are your needs met by the current tooling and processes?

In the pretesting survey this sets a baseline. In the post-testing one, the question directly played into how fit for purpose the respondents found Code Corral to be.

Code Corral General

These questions were only included as part of post-testing, as it was only after having been exposed to the software that a participant could reasonably proffer their views on it.

Was the software installed for you or did you install it yourself?

If the respondent was able to install and configure the software themselves it would signify the ease of use that process offered. It would also act as a spring board for knowledge of what the system constituted. This would allow them to provide answers that showed insight into the various components.

Did you find the software easy to use?

A direct question with a simple "yes" or "no" answer. This question was one of the most revealing questions with respects to the purpose of the survey.

Did using the software impact your access to the source code?

This binary question was asked with the sole purpose of leading participants down one of two possible paths. Either they experienced the effects of Code Corral (rightly, or wrongly) or they saw no effect. The former would allow an opportunity for further questioning, while the latter would establish that (at least in this case) there was no negative effect perceived. This would be a very favourable outcome from an *Availability* perspective.

Code Corral Impact

These questions were asked to determine whether in terms of accessibility the developers felt that the use of Code Corral negatively impacted the ease with which they accessed the source code. They also looked at whether they felt that it impacted the *Confidentiality* in terms of the protection of the source code.

Were you online and able to access the internet at the time?

To use Code Corral the developer would need solid and stable internet access in order to be able to authenticate against the Code Corral server. There are three outcomes to this question:

1. the participant was online;
    (a) but unable to access Code Corral for another reason; or
    (b) Code Corral was unable to determine access despite being active; or
2. the participant was offline and failure is expected.

If the participant is off-line then we expect failure to happen. The requirement for Code Corral is that the developer has access to the Code Corral servers whenever they need to authenticate to gain access again. This can be seen as a possible shortcoming of the system, but it is a known and expected one. The first two scenarios speak to reliability of the solution and to robustness of it respectively.

Did you feel that Code Corral gave you a clear reason for blocking your access?

This question determines whether all reasons for not giving a developer access were properly managed within the software and whether the participants could clearly understand why they were denied access.

Did you feel that Code Corral was right in denying you access to the code?

This question tries to determine whether there are circumstances where the participants were denied access wrongly and not for an accurate reason.

Code Corral Usefulness

These questions are designed to determine whether the developers find the Code Corral system to be useful on a scale of **Not as all useful** to **Very useful**. A favourable response would indicate that Code Corral may have a real place in an enterprise.

How would you describe Code Corral's usefulness?

There is also the opportunity for the participants to comment, giving additional feedback in a free field.

How would you describe Code Corral's effect on your ability to work?

The value of Code Corral is also measured in terms of the effect it had on the participants' ability to perform their work freely. There is a range of options by which to rate this. The responses provided in this section are most significant. It will determine whether the use of Code Corral affected the *Availability* to the source together with the ease with which the participant was able to complete their work.

It is more favorable if the participants response indicates that it did not affect their ability to work or that it made their ability to work easier. This would establish that Code Corral was a useful tool to the participants performing their daily coding duties.

Code Corral Negative Impact

These questions pertain to the negative experience which the participant had with Code Corral. The questions are intent on investigating the underlying causes as well as exploring the possible solutions to any issues.

As an estimated percentage of the planned sprint points, how much do you think that Code Corral impacted your ability to work?

This query is a follow up which determines whether Code Corral's negative effects impacted the participants ability to conduct their everyday work. It adds a value to how the participant felt it negatively impacted their ability to get through their normal work on a scale from *Less than 10%*. The result would indicate that Code Corral had only a minor impact — through to *More than 90%* which would imply that it had greater adverse effect on their ability to work.

What do you think could be done in Code Corral to lessen that impact?

This question will provide further development efforts on Code Corral with something to work on in order to improve issues that led to instances where Code Corral negatively impacted the participant.

Code Corral Positive Impact

These questions determine which features of Code Corral were most appealing to the participants of the survey.

Which feature of Code Corral made working easier for you?

This field is free form to allow the participant to give their honest and objective thoughts on which element(s) of Code Corral made their work easier. This will aid the researcher in determining the effectiveness of each element of Code Corral and the impact they had on the participants.

What do you think could be done in Code Corral to make it more useful?

This question is another free form answer field for the participants to offer their sentiments on what could be done to improve make Code Corral. This information will be used to guide future development efforts to enhance on the current Code Corral system.

Follow-up Questions

These questions aim to gauge the willingness of the participant to be interviewed on their experience of Code Corral. This offers them the ability to discuss their experience in a one on one setting - providing valuable in-depth insight to their experience of the software.

Security Experience

The security questions draw attention to any past experiences the participant had with cyber security and incident response.

What are your views on cyber security?

The participant is offered a free form field to answer this question and will allow them to share any views they may have on cyber security. It is expected to be a wholly subjective answer and will likely only be useful if the participant has agreed to follow up questions discussed above.

Have you ever experienced what you would describe as a security incident or breach?

This question determines whether moving to the next question to gather details about previous knowledge of breaches is useful.

Security Breach Information

These three questions were purposefully conceived to ascertain whether the organisation, for which the participant worked or is currently working for, has experienced what they would define as a security breach or incident. This gives the surveyor context to the answers given in the sections that both follow and precede this question.

Was code lost as a result of the breach?

There are simply two possible answers to this question; either *yes* (the organisation has lost code during and incident) or *no*. This is to determine whether any of the participants and their organisations have lost source code, putting more context around their views on Code Corral. There is also the option for participants to comment in a free form field which will provide valuable context and further insights.

Was there financial loss or damage to the organisation's reputation as a result of the breach?

When organisations suffer loss of data in a breach or security incident there are often financial and / or reputation damage incurred. The options given to the participants allows them to select one, both or none and provides yet more context to the answers they have given.

Where you held responsible for the breach?

This question provides insight into whether the organisations with which the participant has had experience, assign blame for source code breaches; or incidents where source code was lost, to individuals. This will likely colour the perspective of the participant and therefore their views on Code Corral.

Security Stance

Management level survey takers are given a with a brief overview of the CIA Triad. They are afforded the opportunity to indicate which of the three corners of that triangle they deem most significant. As a ranked answer, there can be no ties between the preferences and a clear prioritisation is forced. This ordering preference will then be used to balance the selections made in the scenarios question which follow.

Scenarios

The scenarios described are proposed modes of testing for Code Corral and align with the use cases defined during the project. The intention is to get the participant to rank

the scenarios which increasingly provide *Confidentiality* for the source code at the cost of *Availability* for developers. It highlights the fact that Code Corral is able to be tuned to meet the requirements a company may have of the system.

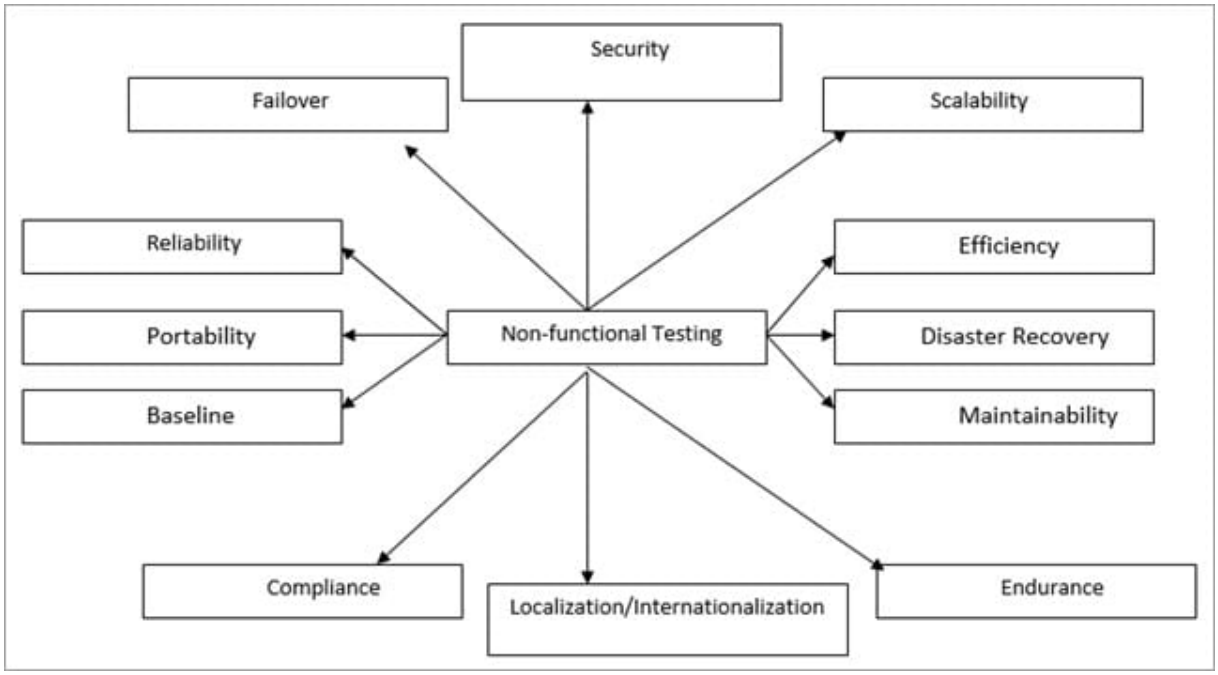


Figure 4.1: Non-functional Testing Overview (Shinde, 2019)

4.2 Non-functional Testing

The term non-functional testing refers to a testing methodology that is performed to test the non-functional requirements given to a system. The criteria used in this methodology of testing determines whether a system is ready for the intended production usage. The non-functional testing requirements focus on the quality of the product rather than its functionality. In this context, the non-functional testing determines whether the software is suitable for the purpose for which it was designed. Consequently, the criteria being tested involves user perspective questions such as "does the software perform in the manner it was designed?", "does the software scale as well as expected (enterprise-level, telecom provider level, internet-scaled)?", "is the the security of the application sufficient?" and "how does the software behave if certain constraints are introduced?".

The following components are used to test the non-functional elements of software:

1. Baseline Testing (section 4.2.1); and
2. Operational Readiness Testing (section 4.2.2);

This type of testing has a more significant influence on customer and user satisfaction than any technical testing may have. The focus was on what would aid in the testing of this proposed software solution in this research.

4.2.1 Baseline Testing

Baseline testing determines whether the baseline criteria are met as set out in the design documentation. This is to establish how software functions at its expected baseline and what those values would be. This testing is not done for specific functions of software but the overall performance parameters of the software application. This is a bench-marking test. The bench-marking results are what are used to improve on. These form the baseline as to how the software will function.

The important factor is to identify which criteria the software should meet, and then measure the software behavior in terms of these specifications which will create the baseline. The significance of baseline testing is that it validates the extent to which the software meets the required baselines in its own performance.

4.2.2 Operational Readiness Testing

Operational testing is performed at the final stages of the software development pipeline and is used to determine whether the build is ready for live deployment. The operational readiness tests stress the baseline, taking a deviation from the norm approach to testing and see how the system performs under various forms of load. Some of the parameters that may be tested to determine the operational readiness of software include:

- **Database backup** - does the database provide the ability to backup the data when any disasters or system errors occur? This test will validate whether the database backup is successful without any data loss.
- **Database recovery** - the database recovery test should be performed to determine if data loss occurs whether the database has the ability to recover from that and to restore lost data. This is an important element of an application, the ability to restore and recover after a failure occurs.
- **Software Installation and Configuration** - Establishes whether the installation and configuration of the software can be done successfully. Each step should be concise and easy to follow. This test examines whether the deployment packages, scripts and configurations work as described in the installation instructions. This verifies the installation and configuration protocols.

- **Rollback test** - Determines whether the software can rollback to a previous working state if failure occurs.
- **Failover** - Verifies whether the software has the ability to proceed as normal when a redundant component fails. It determines the result and effect on the system as a whole if one component fails.
- **Supportability** - The supportability test takes into account the findings of the other tests mentioned above. It concerns itself with how the application will impact the operational teams and how much (manual) effort may be required from them to maintain and provide support to the system.
- **Security Testing** - This test is performed to determine whether the application meets certain security requirements. The tests often performed include things like Penetration Testing, Access Control testing and data protection tests.

Chapter 5

Results

This chapter examines the results of the testing methodology discussed in Chapter 4 to determine whether the proposed Code Corral system did indeed address the concerns raised in the problem statement by helping to bring about a balance between *Confidentiality* and *Availability* of code in the realm of software development. The objective of this research as set out in section 1.3 was used to determine the success of the research.

The Code Corral software was tested functionally to determine whether or not it the passed all the suggested tests in the sections dealing with Integration Testing (4.1.2) and System Testing (4.1.3).

With regards to the testing described under User Acceptance Testing (section 4.1.4), no results are present in this chapter. Instead, a brief explanation as to why those results are missing is provided.

This chapter will discuss the results from the functional testing that could be achieved however, concluding with a summary in which the overall results will be discussed.

5.1 Integration Testing

5.1.1 Access Control

The tests described in listings 4.4 through to 4.7, that are found in section 4.1.2 include authentication stories from both the administrator's perspective as well as the developer's. For the sake of recall they have been included here below:

Access Control Test 1

```
As an administrator of Code Corral
-----------------------------------------------------------
Scenario: "Authorised user clones a repository"
  Given I have granted a user access to a repository
    When they attempt to clone the repository
      Then they are granted access to clone it
```

Access Control Result 1

This test was executed in combination with Test 2 described below as is it proved quicker and easier to actually attempt a real cloning of a repository than it was to simulate it. During the test execution of Test 2, the server logs where monitored along with the database, and the results catalogued. The test was consistently successful: when access was permitted to a given repository then the server would generate and store a new encryption key for the user, for the project selected (`veracrypt-builder`) into the `user_table`, and log:

```
Cloned project veracrypt-builder
```

Result: PASS

Access Control Test 2

```
As a developer using Code Corral
-----------------------------------------------------------
Scenario: "Authorised user clones a repository"
  Given I am granted access to a repository
    When I clone the repository
      Then Code Corral client starts the clone function
```

Access Control Result 2

After ensuring that the public key was correctly associated with the repository and that the client did meet the criteria for that repository, the server granted permission to clone and executed the command:

```
/opt/code-corral/bin/veracrypt --create --volume-type=normal --size=1000M
    --encryption=AES --hash=SHA-256 --filesystem=ext3 --password="2d931510-
    d99f-494a-8c67-87feb05e1594" --pim=1 --keyfiles='' --random-source=/dev/
    urandom "/home/rewtd/.code-corral/veracrypt-builder.hc"
```

That command is taken from a Linux machine and would appear differently on the various operating systems (OS). The values are provided in various ways, they are:

- derived from the Code Corral client's OS defaults (`-filesystem`, and `-random-source`);
- taken from the OSQuery defaults (`-encryption`, `-hash`, and `-pim`);
- values generated for this specific instance of this command (`-password`, a GUID generated server side and passed to volume creation); or
- from values stored as part of the project (`-size`).

When the above command executes the client side logs show:

```
Creating new encrypted volume for veracrypt-builder
Formatting new encrypted volume for veracrypt-builder to ext3
Mounting new encrypted volume for veracrypt-builder at
/home/rewtd/src/veracrypt-builder
```

Result: PASS

Access Control Test 3

```
1  As a developer using Code Corral
2  ---------------------------------------------------
3  Scenario: "Authorised opening a repository"
4    Given I have cloned a repository and have access
5      When I open the repository
6        Then access should be granted
```

Access Control Result 3

After cloning the repository and then closing it, the researcher initiated the `project open` command to the Code Corral client. This is done via the CLI:

```
code-corral project open veracrypt-builder
```

On the server nothing had changed and so when the client requested that the repository be opened for writing again the server had the client initiate the commands:

```
/opt/code-corral/bin/veracrypt --change --pim=1 --keyfiles='' --password="2
    d931510-d99f-494a-8c67-87feb05e1594" --new-pim=1 --new-keyfiles='' --new
    -password="62936e70-1815-439b-bf89-8492855a7e6b" --random-source=/dev/
    urandom "/home/rewtd/.code-corral/veracrypt-builder.hc" 1>/dev/null
/opt/code-corral/bin/veracrypt --password="62936e70-1815-439b-bf89-8492855
    a7e6b" --pim=1 --keyfiles='' --protect-hidden=no --filesystem=ext3 "/
    home/rewtd/.code-corral/veracrypt-builder.hc" "/home/rewtd/src/veracrypt
    -builder"
```

Line 1 of which changed the decryption key for the volume (to match the updated key on the server) and line 2 used that key to open the volume. The change to the decryption key happens so that the client is never actually in possession of the key needed to unlock the volume except when the decryption happens, avoiding the possibility of attacks on the key cache stored locally. It then logged on the client side:

```
Recrypting project veracrypt-builder
Opening project veracrypt-builder
```

Result: PASS

Access Control Test 4

```
    As an administrator of Code Corral
    ---------------------------------------------------------
    Scenario: "I revoke access to a user and they attempt to open"
      Given access was possible to repository
      But it has now been revoked
        When a user attempts to open the repository
          Then the user should be denied access
```

Access Control Result 4

This test was once more initiated by having a user actively attempt to open a volume after access was revoked to the repository. As expected, the client side process for opening a volume was followed and the request as passed to the server. On the server the check for access to the project resulted in:

- the user being denied access;
- them being presented with an error result `400 Bad Request unable to open this project`; and
- the following being logged to the `user_log` for the account used:

```
Attempted to access unauthorized project veracrypt-builder
```

Result: PASS

5.1.2 Data Collection

The next batch of tests described in listing 4.8 through to listing 4.10, in section 4.1.2 included retrieving data from OSQuery from the Code Corral client. The tests have, once again, been included here again for ease of reference.

Data Collection Test 1

```
1  As a developer using Code Corral
2  ---------------------------------------------------
3  Scenario: "OSQuery of a value succeeds"
4    Given that the client is correctly installed
5    And I've asked for access to a repository
6      When I received a request from to look up a value using OSQuery
7        Then OSQuery should return the value requested
```

Data Collection Result 1

When a developer requests access to a repository and passes the first two hurdles (the project they requested exists, and they are authorised access) then their attempt to open or clone the project is logged. Following that, they are asked to execute the queries associated with the project.

There may be more than one query for a project and if that's the case, then they will be asked each query one at a time, together with the query ID and each one is logged, executed and the results returned to the server. The commands executed locally are:

```
1  data=$(echo "select platform,major,minor from os_version;"|/opt/code-corral
       /bin/osqueryi --csv|tail -n 1)
2  curl --request "POST" --silent --output /tmp/tmp.Jes6ReZZJj -w '%{http_code
       }' \
3      --header "X-Code-Corral-Email-Address: grant@ongers.net" \
4      --header "X-Code-Corral-Client:
5      366421f41ea22157342104f32b4bf80d362791ce5c4a9bc9390fdb6d924de235" \
6      --header "X-Code-Corral-Tool-Veracrypt:
7      4b0dbdbb700c8ce8ccdd5121ee50da3da7b43ca8152a67112d864605650bdbe1" \
8      --header "X-Code-Corral-Tool-OSQuery:
9      4b0dbdbb700c8ce8ccdd5121ee50da3da7b43ca8152a67112d864605650bdbe1" \
10     --header "X-Code-Corral-Auth:
11     IDiqcXxIxsnsoBVjOwoq5JLUnVnbeLK6le94/gyOmXBbU8oAlzjci9Bu+jM5yV54b/
12     MpY/0bllvc2lzpphplS5ONDzTsbzOGhxkA2yP+H88fhb2mNstam2B61ERgm9G6HJvs
13     +yXwmG4ngB/RpVyRClnYj2syQ4efP0Ja8uc2eVHJPcAeiQDmFWQ77utsI9f5fj3H4I
14     hpJ33Pe6aTGHmc3aoq8r4q/j1spivIQ5Hy8UdsiBCkgdA0GOQUYywpoU3w2kJprQ2l
15     /fXKbQw+6u8Ir1SVqW7YMQr/TCuCNaaKzwLlsIdjnhVmmD0pZXDsRjQoebFSQ6IIcN
16     oaYkh4UA315g==%" \
17     --data "ubuntu|18|4" \
18     --stderr "${error_log}" "https://code-corral.herokuapp.com/osquery/
       answer/3"
```

Lines 3 through 7 contain client populated variables for the header added to `curl` (which are added to each transaction) and line 8 the data which is the results of the query executed in line 1 (`ubuntu|18|4`).

And the local log will show the following entry:

```
Answering remote host query with ID:3 about local system
```

Result: PASS

Data Collection Test 2

```
As an administrator of Code Corral
-----------------------------------------------------------
Scenario: "OSQuery of a value succeeds"
  Given a request triggered an OSQuery lookup
    When the data is returned
    And it matches the required value
      Then the server should return SUCCESS
```

Data Collection Result 2

Continuing the story from the first test, when the values are returned to the Code Corral server and logged:

```
ProjectOSQuery(2) Answer:  ubuntu|18|4
```

If they match the expected result then the client is sent a response:

```
200 osquery|ack
```

And the system will execute the `clone` or `open` actions as required.

Result: PASS

Data Collection Test 3

```
As an administrator of Code Corral
-----------------------------------------------------------
Scenario: "OSQuery of a value fails"
  Given a request triggered an OSQuery lookup
    When the data is returned
    And it does NOT match the required value
      Then the server should return FAILURE.''
```

Data Collection Result 3

Should the required result not match however then the system logs as above and returns the user an error result of:

```
400 Bad Request must match project query 3
```

Result: PASS

5.1.3 Volume Mount / Dismount

Volume Mount / Dismount Test 1

```
1  As a developer using Code Corral
2  ---------------------------------------------------
3  Scenario: "Cloning a repository succeeds"
4    Given a repository is not yet cloned locally
5      When a clone request is sent
6      And access is allowed
7      And cloning permitted
8        Then VeraCrypt should create and mount a volume for this purpose
```

Volume Mount / Dismount Result 1

This test was previously executed as a part of the tests performed in section 5.1.1, during Access Control Test 2. When the repository was successfully cloned the logs locally showed:

```
Mounting new encrypted volume for veracrypt-builder at
/home/rewtd/src/veracrypt-builder
```

Result: PASS

Volume Mount / Dismount Test 2

```
As a developer using Code Corral
---------------------------------------------------------
Scenario: "Volume mount succeeds"
  Given a locally cloned repository
    When an open command is requested, authorised and executed
      Then the volume should successfully mount
```

Volume Mount / Dismount Result 2

As before, this test was executed as part of the tests on Access Control (Test 3) where a previously cloned repository was opened. Local logs showed:
`Opening project veracrypt-builder`

Result: PASS

Volume Mount / Dismount Test 3

```
As a developer using Code Corral
---------------------------------------------------------
Scenario: "Volume dismount succeeds"
  Given a locally cloned repository
    When the close command is requested
    And it is executed
      Then the volume should successfully dismount
```

Volume Mount / Dismount Result 3

This test verifies that Code Corral is able to dismount volumes that it mounted in the first instance. There are two variants of this test, as discovered during the testing itself. One of those two is when there are open files held by an integrated editing environment (IDE) or another process in the system, and the other is when all files are closed and inactive.

The command executed by Code Corral is the following:

```
/opt/code-corral/bin/veracrypt -d "/home/rewtd/.code-corral/veracrypt-
    builder.hc"
```

During the testing the researcher discovered that in some circumstances this command will fail if there are open file handles held by the operating system.

This can be corrected by passing `-force` to the command-line used[1] which forces the dismounting of the volume despite there being open file handles. Doing this however, requires more rigorous testing as it is possible this might leave the volume is an unusable state.

It's also possible that using the same switch on mounting the volume would result in the usage on dismount being less destructive.

Result: INCONCLUSIVE

[1] https://www.veracrypt.fr/en/Command%20Line%20Usage.html

5.2 System Testing

5.2.1 Deploy a Server

Test

```
Feature: "Installing a Code Corral server"
As a administrator of Code Corral
-------------------------------------------------------
Scenario: "1. Set up the DBMS"
  Given a Linux server with PostgreSQL
  (or a cloud provider with a managed RDMS solution)
    When all the scripts in ./sql are executed
      Then the database schema should match the models in ./models
-------------------------------------------------------
Scenario: "2. Set up the web-server"
  Given a Linux server with Ruby
  (or a cloud provider with Ruby)
    When dependencies are installed
    And database migrations run
    And deployment complete
      Then the web-server should return "no email" to a GET request:
      from "https://code-corral.herokuapp.com/api/1.1"
```

Result

GitOps[2] is a term that describes a solution to manage operations by means of Git using a CI/CD pipeline driven by changes made and committed to Git. Owing to the optional step included this the approach, it allows deployment from Git commits to happen.

There are still a few manual steps to deploying a Code Corral server for one's own purposes. Future work (section 7.3) would include taking Code Corral to other cloud providers. The manual steps required for Code Corral at present are as follows:

Set up a Heroku account and deploy a new application (figure 5.1)

Configure Heroku for PostgreSQL (figure 5.2)

Configure Heroku for automatic deployment (figure 5.3)

[2] https://www.weave.works/blog/gitops-operations-by-pull-request

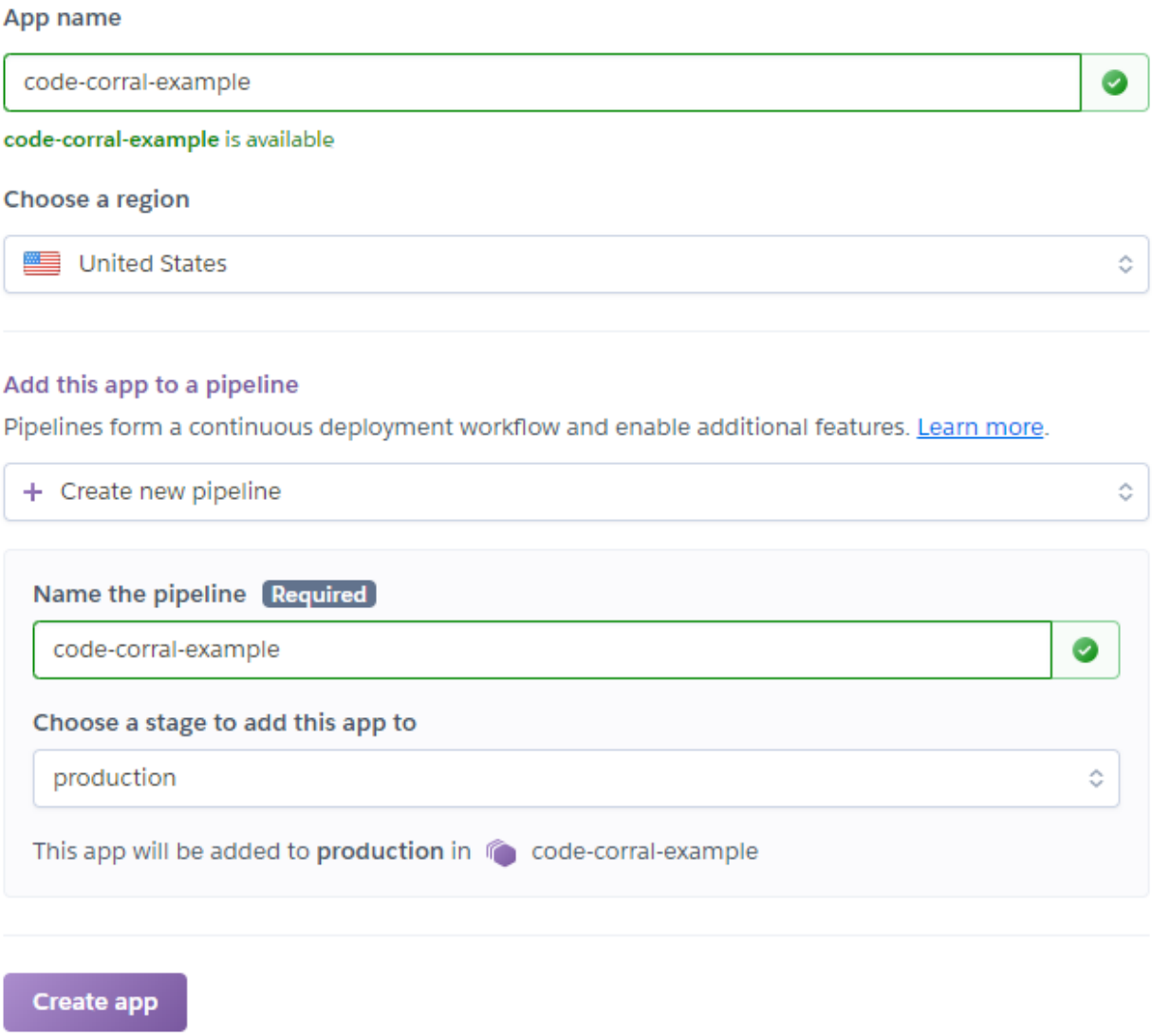


Figure 5.1: Deploy an Application in Heroku

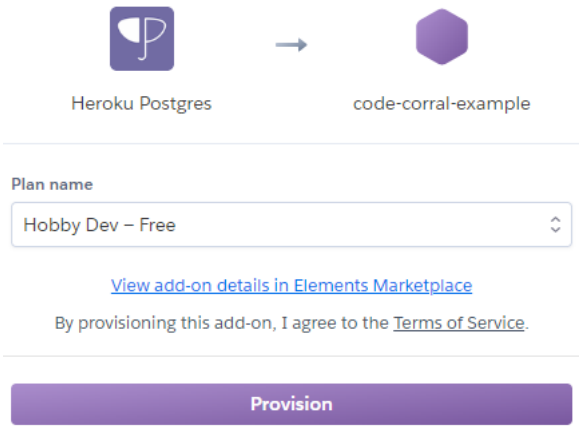


Figure 5.2: Configure Heroku for PostgreSQL

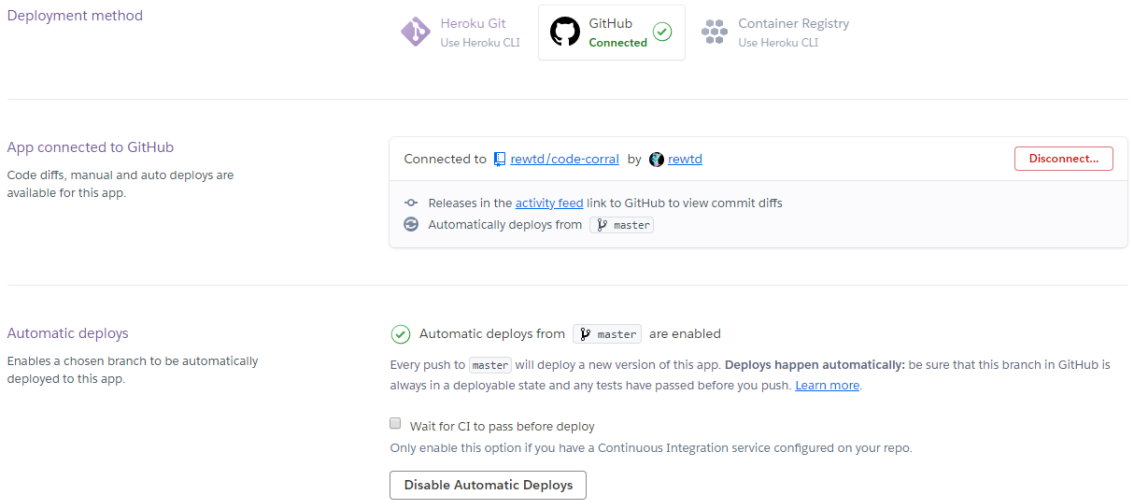


Figure 5.3: Configure Heroku for Automatic Deployment

All that then remains is to deploy Code Corral (or to push a change to the branch configured for automatic deployment) and then the Code Corral server should be deployed automatically.

Testing at this point entails hitting the API at:
`https://code-corral-example.herokuapp.com/api/1.1` (per the example)

This should currently return an error:
`400, no email` (The server has not been configured just yet).

Result: PASS

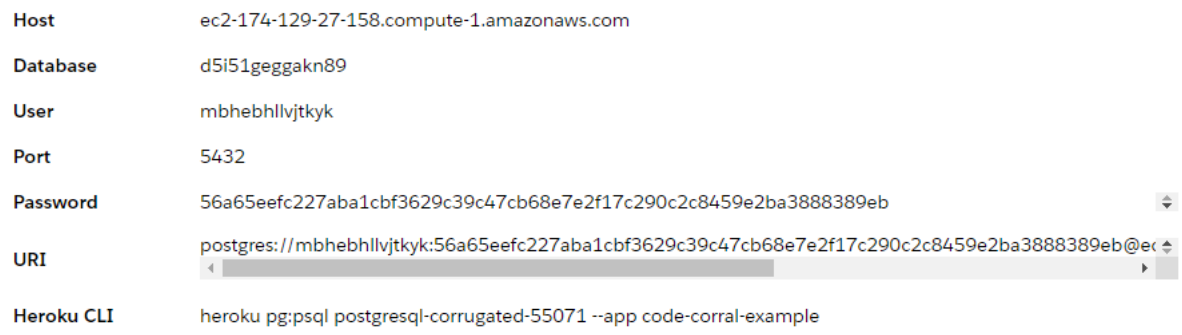


Figure 5.4: Heroku PostgreSQL Credentials

5.2.2 Configure a Repository

Test

```
Feature: "Installing a Code Corral server"
As a administrator of Code Corral
-------------------------------------------------------
Scenario: "1. Add users"
  Given an installed Code Corral server
    When a database client is connected to the DB
    And user details are added
      Then the database should store those values
-------------------------------------------------------
Scenario: "2. Add projects"
  Given an installed Code Corral server
    When a database client is connected to the DB
    And project details are added
      Then the database should store those values
-------------------------------------------------------
Scenario: "3. Add rules (queries)"
  Given an installed Code Corral server
    When a database client is connected to the DB
    And query details are added
      Then the database should store those values
```

Result

Connecting a GUI to the database on Heroku requires that the credentials for the system access is being sought to are obtained (see figure 5.4). The tester then connects the pgAdmin interface and needs to:

Data Output | Explain | Messages | Notifications

	id [PK] integer	email text	current_encryption_key text	created_at timestamp with time zone	active boolean
1	1	clive@cr...	[null]	2017-10-29 10:31:31.981529+00	true
2	2	grant@o...	[null]	2017-10-29 14:23:44.953749+00	true

Figure 5.5: Add Users to Code Corral

Data Output | Explain | Messages | Notifications

	id [PK] integer	name text	repository_uri text	recommended_disk_space_in_mb integer
1	1	code-c...	git@github.com:re...	1000
2	2	veracry...	git@github.com:re...	1000

Figure 5.6: Add Projects to Code Corral

Add users to Code Corral (figure 5.5).

Add projects to Code Corral (figure 5.6).

Setting up queries is much the same process (see figure 5.7), and there are example queries one might want to consider making use of in Appendix C.1.

The system accepts the data correctly and one can verify that the configuration is complete and functional by executing the integration tests laid out in section 5.1 or the system test defined in section 5.2.4 below.

Result: PASS

Data Output | Explain | Messages | Notifications

	id [PK] integer	query text	valid_for interval
1	1	select version from kernel_info;	10 days
2	3	select platform,major,minor from os_version;	10 days
3	2	select computer_name from system_info;	10 days

Figure 5.7: Add Queries to Code Corral

5.2.3 Install Client

Test

```
Feature: "Installing the Code Corral client"
As a developer using Code Corral
-------------------------------------------------------------
Scenario: "1. Fetch to Code Corral client"
  Given an installed Code Corral server
    When the install command is executed
      Then the Code Corral client should execute
-------------------------------------------------------------
Scenario: "2. Install Curl"
  Given curl is not installed
    When Code Corral client installs
      Then curl should be installed
-------------------------------------------------------------
Scenario: "3. Install VeraCrypt"
  Given veracrypt is not installed
    When Code Corral client installs
      Then veracrypt should be installed
-------------------------------------------------------------
Scenario: "2. Install OSQuery"
  Given osqueryi is not installed
    When Code Corral client installs
      Then osqueryi should be installed
```

Result

Installing Code Corral on a fully supported system is as simple as executing the two commands given in the README.md file:

```
wget -qO- https://code-corral.herokuapp.com/client/bash/code-corral | bash
ssh-keygen -e -m pem -f ~/.ssh/id_rsa | openssl rsa -RSAPublicKey_in -
    pubout
```

The first line initiates the installation, the second was required to produce the `ssh` public key needed when setting this user up in Code Corral on the server in test 5.2.2.

Code Corral, run on a system that supports it, will always first ensure that `curl` is installed and available to the logged on user by executing `which curl`. If it does not find

it installed then it will make use of `apt-get` to install it. This would require `sudo` access as it runs this install as root. It will then set up the folders required for itself and ensure that a local copy of the `code-corral` client is deployed.

The client will then ensure that VeraCrypt is installed by looking for the Code Corral version of the binary which is usually deployed to `/opt/code-corral/bin/veracrypt`. If the file is not found there (or if the hash of the version there does not match the expected version in Code Corral) then `curl` is used to fetch the appropriate binary for the OS running from the Code Corral server. Root access is again required to place the binary in the correct location.

Finally it will ensure that OSQuery is installed following the same steps as it does for getting the `osqueryi` binary as it did for the `veracrypt` one.

Logs for this process on the client will look something like this:

```
Curl is a required dependency. Installing ...
Installation complete, please add /opt/code-corral/bin to your PATH environment:
   export PATH=/opt/code-corral/bin:\$PATH}
   after which you will be able to use code-corral
Veracrypt is a required dependency. Installing ...
OSQuery is a required dependency. Installing ...
```

Result: PASS

5.2.4 Checkout a Repository

Test

```
Feature: "Using the Code Corral client"
As a developer using Code Corral
-----------------------------------------------------
Scenario: "Clone a repository"
  Given a configured Code Corral server
  And an install Code Corral client
    When a clone command is executed for a repository
    And the developer has access to that repository
    And meets the requirements for access
      Then a volume should be created
      And the volume should be mounted
      And the repository should be cloned into the volume
-----------------------------------------------------
```

Result

This test was for all intents and purposes completed as part of integration testing in section 5.1.1, in Test 2. When access was permitted and the repository was cloned successfully and the logs showed the following:

```
Creating new encrypted volume for veracrypt-builder
Formatting new encrypted volume for veracrypt-builder to ext3
Mounting new encrypted volume for veracrypt-builder at
/home/rewtd/src/veracrypt-builder
```

Result: PASS

5.3 User Acceptance Testing

At the commencement of the research an organisation was approached and gave their consent to have the survey distributed to developers within their corporation. The Code Corral tool was to be introduced within their environment and the User Acceptance testing to be measured. Owing to constraints experienced by the company ascribed to development within their environment and the time constraints associated with the completion of a half thesis the consent and *Availability* was withdrawn by the organisation. As explained in section 4.1.4 with regards to ethics within a study participants can withdraw their willingness and consent for involvement. This unfortunate circumstance came to pass — and that at an advanced stage of this research. It was therefore decided to not include the results for the survey and endeavour to test this at a later stage.

Chapter 6

Discussion

This chapter puts forward the research in terms of both the parallels and differences between the various solutions available to organisations, reviewed in chapter 2, and in terms of how those solutions meet the requirements set up in the problem statement in section 1.2.

The chapter commences with section 6.1 which compares the parallels and differences of the solutions examined in turn. These solutions include Network Access Control (NAC) which was examined in section 6.1.1 It goes on to further compare this solution — and specifically the NAC Gatekeeper — with the proposed solution designed by the researcher (Code Corral) in section 6.1.1. The section concludes with section 6.1.2 by exploring the future of Network Access Control (NAC) within organisations.

The chapter proceeds in section 6.1.3 to compare the DLP solutions which were discussed in chapter 2.4 with Code Corral and to draw conclusions about their compatibility with the problem statement.

Furthermore, in this chapter, the discussion surrounding the Zero Trust implementation from Google BeyondCorp was explored in section 6.1.4.

Git-related solutions where presented the in section 6.1.5 to determine to what extent they address the CIA triad in terms of developers. The problem statement and Git solutions were contrasted with Code Corral.

To conclude the chapter Code Corral itself is reviewed in terms of the problem statement. A summary of how it meets with the goals described in the problem statement is presented with a strong focus on how these solutions address the balancing of the CIA triad.

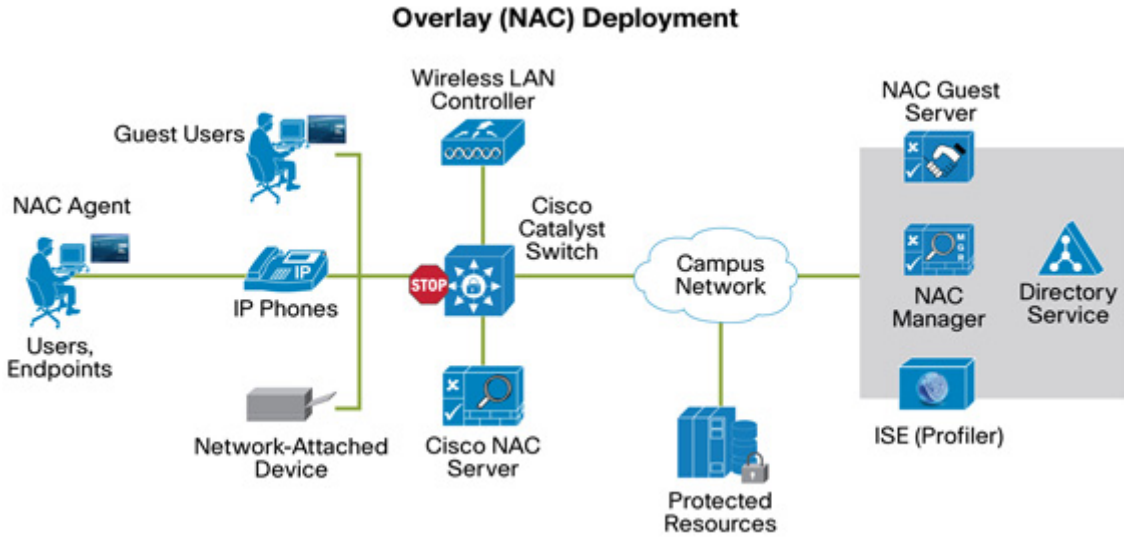


Figure 6.1: NAC Overlay (Cisco, 2014)

6.1 Parallels and Differences

6.1.1 Network Access Control (NAC)

NAC attempts to facilitate access to resources on the network by a verification process that involves confirming that the device attempting to connect to the network meets the criteria defined for access (Tang and Liu, 2007). It achieves this by having the NAC enabled agent on the device provide it with information about the system on which it is running. At times this agent ships as part of the operating system, as was the case of Microsoft's tooling. It could also be an additional piece of software that ships as part of the solution, as in the case of the Cisco client reviewed.

The NAC service is based on a client server model. In fact, there are a number of services that the Cisco solution mandates; see, for example, the Cisco NAC Overlay in figure 6.1, including a NAC Guest Server and NAC Manager in addition to the Cisco NAC Server. This client server model divides the responsibilities for the assurance between the two components and the consequence is that if a client is unable to provide information about its state, the server is unable to authorise its attempt at connectivity. Typically, as can be seen in the case in figure 6.1, there is an alternative network — a virtual local area network (VLAN) — to which those devices are directed. The aforementioned network offers only services required to correct the issues on the device, whether they are related to the client agent not being present or functioning correctly, or another issue on the client

which the client agent reported as not being compliant (Microsoft, 2016).

The client agent on a device will review some aspects of the device's "health", for instance the status of anti-virus (AV), the current patch level, or the device firewall state (typically a combination of these and many others) and then pass the results of that review to the NAC server component. Subsequently, the server then determines whether access is granted or the devices needs to be relegated to the remediation network.

If the latter is the case, it signifies that the device being denied access will not be denied access to the whole system. Normally the remediation network will host services to allow a device to upgrade its state, and possibly access to the internet. In the case of Code Corral, access denial is specific to the resource requested, but the client still has access to resources that may be able to remedy the issue so that it can attempt to connect again. Any errors (Appendix B.2) or warnings (Appendix B.3) encountered are output to the console and access is not denied to the network or anything else, other than the code store itself.

Similarities between the NAC gatekeeper and Code Corral

In the context of network access, the gatekeepers are the management tools used to allow others access to information (Antonelli, Link, and Metcalfe, 2013). Broadband providers' networks act as a platform towards the internet ecosystem participants', to communicate using the broadband provider as a gatekeeper proxy (Commission, 2015). In a similar fashion, the Code Corral will act as a gatekeeper proxy that allows access to the Git repository. Rhodes-Ousley (2013) states further that gatekeepers

> "... provide(s) intelligence and control certain routing and authentication, authorization, and accounting (AAA) security functions [...] [It] can also perform and assist with certain types of address translation, and can consolidate administrative control elements such as call detail records (CDR), communication with monitoring and management systems..."

In practical terms, this translates to having a guardian watching traffic move through the entrance to a system and then ensuring that the traffic is directed (or redirected) in the right direction so that only traffic destined for secure areas, that meets the security requirements, reaches that point. The rest end up in areas where their security issues can be resolved or where they can do no harm.

The Code Corral software tool proposed in this paper for securing software development by means of developer access similarly implements criteria that has to be met in order for a developer to access code repositories. This acts as an additional layer of security at the point of access.

Distributed version control, by definition, eschews centralized access controls. There are no fixed rules built into Git which can assist companies with controlling access to their code. Git is just a content tracker: it does not double as a gatekeeper or access control manager (Westby, 2015). For the purpose of this research, the proposed use of Git is purely in relation to version control. Git is used as the repository for tracking changes to as well as storing the code. The implementation proposed to act as a gatekeeper and proxy is Code Corral. This is an excellent fit because Code Corral implements its own security controls and layers which developers need to comply with in order to gain access to the code in a secure manner.

6.1.2 Future of NAC within Organisations

While NAC is a useful comparison for the work in this thesis, in that it is a form of access control that allows companies to define the requirements for access, there are some concerns as to the future of these solutions. NAC relies heavily on the fact that there is a "safe" network and strong perimeter defences, but this world view is changing to match more that of the Google BeyondCorp as talked about in section 2.5.

Cisco's Network Admission Control, and specifically the Cisco Clean Appliance (CCA)[1], which is the solution through which Cisco previously provided NAC services has been made End Of Life (EOL). Subsequently, customers are being persuaded to opt for the Network Visibility and Segmentation route instead[2]. Microsoft's Network Access Protection[3] has been phased out and replaced by their InTune client. This offers NAC integrations in Microsoft's answer to mobile device management.[4]. These integrations allow InTune to provide the client half of a NAC solution while Microsoft no longer offers the server side. Interestingly, the very same client Microsoft offers for NAC provides the client side for their Bring Your Own Device (BYOD) strategy, their zero trust computing strategy (Conditional Access)[5], and their Mobile Device Management (MDM) tooling.

[1] https://www.cisco.com/c/en/us/products/security/nac-appliance-clean-access
[2] https://www.cisco.com/c/en/us/products/security/network-visibility-segmentation
[3] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008
[4] https://docs.microsoft.com/en-us/intune/network-access-control-integrate
[5] https://docs.microsoft.com/en-us/intune/conditional-access

6.1.3 Data Loss Prevention (DLP)

There were three solutions examined in terms of the problem statement as set out in section 1.2. All three were found to have particular limitations in their detection of source code in that it is reliant on the correct tagging of information, or based on acceptable document and unacceptable documents and or search criteria. These solutions rely on being configured in such a manner that they can accurately detect confidential source code. However, in code development strings can often be complex. These three solutions can only detect what is known. This may lead to inaccurate detection of source code. The primary limitation of MyDLP and EndPoint Protector is that the context in which the confidential is used is not taken into account. Only Symantec takes this into consideration. There seems to be some difficulty balancing the *Confidentiality* with the *Availability* in terms of DLP systems, specifically for the development pipeline. When *Confidentiality* is achieved it is in favour of taking away the *Availability* of source code for the developer. When the *Availability* is achieved (by making use of a local repository) then *Confidentiality* is sacrificed. Based on the research, these three solutions do not offer a balance in the CIA triad. However, of the three which were examined Symantec offered the most granular approach to DLP in terms of source code.

There is a clear symbiosis possible between Code Corral and the Symantec Data Loss Prevention solution. The latter would allow one to enforce rules around preventing developers from moving the checked out code from the VeraCrypt volume in which it is stored. This could allow for the inclusion of more robust controls for cases where the developers have gone rouge.

6.1.4 Google BeyondCorp Implementations

This particular solution is focused on a more eloquent method of balancing the CIA triad in terms of how data or other resources are accessed. The focus with BeyondCorp is to shift the access controls from the perimeter of the protected environment to individual services by authenticating (and authorising) users and their devices. In the development industry it is crucial for developers to be able to access their resource from virtually anywhere and it is vital for the owners of these resources that it is done in a secure manner. This is where the zero trust network approach succeeds; it allows for individuals to access data from virtually anywhere without having to depend on virtual private networks (VPNs) that effectively would put them inside that perimeter.

The problem statement discussed in section 1.2, is about the balance of *Availability* and *Confidentiality* in the realm of secure code development and trying to find a means which allows for free access but provides better security. This solution tries to eradicate the traditional approach of gated access to services based on physical location and moves it to an approach which allows for access to services from any location. The focus is on the actual user access and the device state. When a device has met the criteria set out in the access rules of the organisation it will gain trusted access to the service or data as long as the user also meets the requirements. The former is managed by the ACE (pulling from the Device Inventory Database), the latter by the Access Proxy (pulling from the User / Group Database) as shown in figure 2.2, in section 2.5. In terms of its usability for securing code whilst in the pipeline, it partially fulfils this need. Google BeyondCorp definitely ticks the boxes for determining access to a specific service or data set if the device and user meet the requirements. This protects the service and/or data sufficiently and adds to *Confidentiality* in terms of who has access to the data and how they gain access to it. If configured correctly, the developer should be able to access the services or data they require.

Google BeyondCorp manages the connection or method used to gain access to service or data. It does not offer any prescribed solution that covers either the encryption for the data remotely or locally. Additional implementations would be needed to achieve this — although, as a rule, BeyondCorp access is either over Transport Layer Security (TLS) or Secure SHell (SSH). What BeyondCorp offers is effectively a secure framework one could use to allow developers to access data lending it more to towards *Availability* of the service and data rather than the focus being on *Confidentiality*.

This would mean that the local repositories would still remain insecure unless the solution also includes additions to it to cover that local storage problem. The focus for Google BeyondCorp is on how users gain access to services and data securely whilst not relying on a corporate network and allowing for more productivity. As the local repository would remain unprotected (not encrypted), it would not aid to the *Confidentiality* aspect of the research. Although this is a novel approach to how users and devices access resources, it is not a solution that can be employed as it stands.

6.1.5 Git-related Solutions

Transparent Git Encryption does not offer a comprehensive solution for the *Confidentiality* of the source code itself or full repository encryption. This solution focuses on

the protection of the data within a file which still leaves portions of files (and the intellectual property they contain) exposed to unauthorised access. It also does not offer any protection for local repositories and focuses on the protection of portions of data whilst in transit and storage remotely.

The **git-remote-gcrypt** solution aids in the push and pull of repositories and encrypt the whole repository using GnuPG. This offers a more secure (*Confidentiality*, in terms of CIA) approach to protect the data contained in the remote repository but offers no solution locally.

The **git-crypt** solution offers the ability for more *Confidentiality* on the remote repositories similar to the other Git-related solutions. However, the more secure or *Confidential* the solution is, the less *Available* the source code is to the developer. The same is true for the converse: by adding more *Availability*, *Confidentiality* is sacrificed and both happen at a fairly steep rate. There is no clear solution within `git-crypt` to balance this, or to make the push and pull less severe. In addition, the local repository remains vulnerable and not secure.

6.2 Viability in the Real World

The process of software development according to (Singh and Gautam, 2016) is a process in which the needs of the user are translated into a software product. The process aims to translate and address the identified needs of the intended user of the product by implementing a design in code. Subsequently, for Code Corral to be viable in the real world, it needs to be examined in terms of whether it addresses a problem and whether it is a usable in real life. The sections which follow will examine the viability of Code Corral in real world applicability terms.

6.2.1 GUI Client Side

When designing software which will be used by people, it is important to note that there are two areas of speciality in play. Namely, (1) Software Engineering and (2) Human Computer Interaction. These are often disjointed as (Shahnaz, 2013) explains. The former involves the system life-cycle and the latter is related to the Graphical User Interface Development Process. A Graphical user interface according to Criado, Vicente-Chicote,

Padilla, and Iribarne (2010) plays a key role in human-computer interaction as it links the system with the end-user and allows for information exchange and improved communication. The concern that users increasingly demand that application have adaptive interfaces that dynamically evolve in response to users' needs should be brought forward as a point of discussion. Meeting the users' demands in this manner would mean providing a graphical user interface with run-time adaptation capabilities.

The addition of a Graphical User Interface on the client side adds to the usability of the system and has a range of functionality which allows the system to efficiently and adequately accomplish certain tasks for users. Shahnaz (2013) states that the actual effectiveness of a system can be measured when there is a proper balance between the functionality of a system and the usability of the system.

In terms of the balance of usability and functionality in Code Corral, it probably needs to implement a Graphical User Interface on the client side. This is not in the current version of Code Corral. The system, as it stands, needs improvement on the usability side of the balance equation.

6.2.2 GUI Server Side

The main purpose of the User Interface on the server side is to make administrative tasks easier. As the database contains data about the project, users and queries it will need to be constantly updated as new developers join or leave or as projects are added or removed. There needs to be an interface through which the organisation can interact with the server. Code Corral, as a solution, is currently in a PoC and for it to be fully viable in the real world a user interface that allows administrators perform these tasks should be implemented and designed.

Code Corral should have a web-based user interface where administrators would be able to make configuration changes to the server side of the solution. This should preferably be facilitated with the assistance of an advanced programming interface (API) that would manage access to the database back-end. Currently Code Corral does not have this ability and everything is configured using generic database clients that write directly to the tables.

6.3 Summary

Various solutions that may be implemented in isolation or in conjunction with other solutions have been identified. It is clear that although some of the solutions examined partially fulfil the requirement of balancing the CIA triad for software development, the main focus of these solutions is either *Confidentiality* or *Availability*.

It has been proposed that Git be used as a repository for the code. The Code Corral solution, specifically designed to address the problem statement in this research, will act alongside Git as gatekeeper and proxy for access to the Git repository. This is a symbiotic process as Code Corral implements security controls and layers which a developer would need to be compliant with in order to be allowed to access the code. Code Corral also leverages VeraCrypt to protect the local repository from device theft. The server side of Code Corral allows for the revocation of keys if a device is stolen or lost. This offers protection in terms of *Confidentiality* of the local repository to a degree not seen in the other solutions and allows *Availability* for the users.

The main remaining predicament with Code Corral is that it does not prevent the developer from copying the code while the volume is mounted. A secondary issue is that the software can have some effects on *Availability* if the developer is not online to authenticate. The real-world effects of connectivity on a development team in using Code Corral could not be properly examined due to UAT not being possible and this is a key aspect that future work must cover to ensure the solution is properly viable.

Although the solution is not perfect, future work will look at addressing and overcoming these challenges.

Chapter 7

Conclusion

This chapter will draw conclusions on the overall state of the research in terms of meeting the research goals, and how it addresses the problem statement as set out in chapter 1.

7.1 Research Summary

The introductory chapter (chapter 1) defined the problem statement, research goals, the scope of the research, and the structure of the document. The aim of the research was to balance the CIA triad for software development and to bring the centre of balance back towards *Confidentiality* of the source code while still affording developers the *Availability* to the source code.

Chapter 2 presented the related work which was examined by the researchers. This chapter covered related concepts such as Google BeyondCorp, DLP solutions, the CIA triad itself, and more related terms that are associated within the software development sphere. The conclusion was drawn that there were no solutions that solved the balance between *Confidentiality* and *Availability* in terms of software development. There was also limited academic literature available on the technical solutions reviewed, consequently, technical white papers were used as references. This identified a gap within the academic field where academic research should be conducted based around viable solutions. Some research brought forward zero trust networking. This is a fairly new practise and is being explored by other researchers.

Chapter 3 discussed the various components that formed part of the design of Code Corral and their implementation in terms of the proposed solution. It discussed both the client agent and the server side software which make up Code Corral.

Chapter 4 is the chapter in which the methodology was set out for the testing of the Code Corral solution. Functional Testing methodology was discussed in terms of the various tests which were conducted. Furthermore, Non-Functional testing was also explored. The chapter's aim was to determine the tests which should be performed in terms of Code Corral.

Chapter 5 detailed the results from the tests methodology which were set out in chapter 4. These results were used to determine whether the Code Corral software was functional and acceptable for use in terms of the problem statement discussed in chapter 1.

Chapter 6 outlines the discussion of the similarities between Code Corral and various other solutions which were considered. This also highlighted the parallels and differences of the various solutions and Code Corral.

Chapter 7 concluded the research with a summary of the findings and the evaluation as to whether the goals of the research had been met and to what extent. This also included hypotheses as to future work.

7.2 Evaluation of Research Goals

In order to be in a position to evaluate the effectiveness of this research, the outcomes discussed within the thesis are compared with the initial research goals which were presented in chapter 1. The research goals were:

1. Identify existing solutions which can and are used to protect IP in the form of source code.
2. Perform a comparative study of these solutions in terms of the problem statement in balancing the CIA triad.
3. Develop a solution that offers a better integration with Git without compromising the versioning control it offers, whilst leveraging a more secure way to do access control of the source code.

4. Develop a solution that allows for the local repository to be protected whilst ensuring that access can be revoked.

5. Maintain the *Availability* for the developer to the source code without impacting productivity.

7.2.1 Research Goal 1

Identify existing solutions which can and are used to protect IP in the form of source code.

This research goal was satisfied in terms of identifying solutions which (partially) protect the IP of source code. The main concern identified is that source code becomes less *Available* when *Confidentiality* is achieved or when *Availability* is achieved then the *Confidentiality* is compromised.

7.2.2 Research Goal 2

Perform a comparative study of these solutions in terms of the problem statement in balancing the aspects of the CIA triad.

In chapter 2 various solutions are compared to determine whether these solutions can balance the CIA triad for protecting source code. The solutions all offer to solve one of the three elements but do not adequately balance *Confidentiality* and *Availability*. A comparison was successfully drawn between the various solutions that are on offer in terms of protection of IP in relation to source code.

7.2.3 Research Goal 3

Develop a solution that offers a better integration with Git without compromising the versioning control it offers, whilst leveraging a more secure way to do access control of the source code.

A new solution was developed for the purpose of this research. It integrates with Git, which is commonly used for versioning control and allows for development teams to continue working as they do under normal circumstances. The researcher further developed

a gatekeeper that controls the access to the source code by using a zero trust approach to access control. This was combined with encrypting the local repository. This solution helps to address the balance between *Confidentiality* and *Availability* in a more stable fashion than any of the other solutions reviewed.

7.2.4 Research Goal 4

Develop a solution that allows for the local repository to be protected whilst ensuring that access can be revoked.

The use of VeraCrypt to encrypt the local repository and the zero trust approach to access allows for the decryption keys to be revoked at any time. This would leave the local repository encrypted, thereby addressing the goal of creating a solution that allows for protection of source code in cases where the local machine has been lost or stolen.

7.2.5 Research Goal 5

Maintain the *Availability* for the developer to the source code without impacting productivity.

This research goal was only partially achieved owing to the fact that full user acceptance testing is required to properly determine how completely this goal is met. In theory, the researcher partially achieved this but in practice this testing is incomplete, with user acceptance testing being left for future work.

7.2.6 Summary of Evaluation

Overall the research objectives were met. However, owing to the user acceptance testing falling through at an advanced stage of the research and it being moved to future work the results of that testing are unavailable.

7.3 Future Work

The Code Corral solution, as it stands at the conclusion of the thesis, is functional. It is only a proof of concept and not yet ready for real world deployment. The goal for future

work is to release this as an OWASP project - allowing other talented and innovative minds to contribute in an on-going fashion. There are still a few outstanding elements before this can be achieved, some of which were discussed in more detail in chapter 6. Some future work which will be continued with are as follows:

- GUI design for the user side needs to be designed, implemented and tested.
- The GUI for the server side needs to be designed, implemented and tested.
- User acceptance testing should be conducted to determine the level to which the Code Corral is accepted by the Software Developer space.
- The repository should be made public for input from other developers.

Before the Code Corral solution can be made available more widely, there needs to be user acceptance testing done to determine the impact it may have on developer's access to source code. Future work includes this user acceptance testing as set out in chapter 4 in section 4.1.4. This can be achieved by allowing an organization to implement Code Corral — which will be freely available on GitHub — in their software development environment for a set duration of time and to complete the included surveys.

Furthering the desire for full GitOps, and looking to extend Code Corral into other cloud providers, future work may include building deployment options that will leverage other cloud platforms like Amazon AWS, Google Cloud Platform and Microsoft Azure,

The DLP solutions reviewed in section 2.4 showed some excellent potential for being compatible with Code Corral. A combination of the two can extend the threat model for which Code Corral was designed. By preventing a developer from copying out the code from the mounted volume, the software's limitation in this regards may be successfully overcome.

References

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. Agile software development methods: Review and analysis. *CoRR*, abs/1709.08439, 2017.
URL http://arxiv.org/abs/1709.08439

Albing, C., Vossen, J., and Newham, C. bash Cookbook. O'Reilly Media, first edition edition, 2007.

Alneyadi, S., Sithirasenan, E., and Muthukkumarasamy, V. A semantics-aware classification approach for data leakage prevention. In **Susilo, W. and Mu, Y.**, editors, *Information Security and Privacy*, pages 413–421. Springer International Publishing, Cham, 2014. ISBN 978-3-319-08344-5.

Antonelli, C., Link, A., and Metcalfe, S. Technology Infrastructure. Taylor & Francis, 2013. ISBN 9781317990574.
URL https://books.google.co.za/books?id=caXaAAAAQBAJ

Ayer, A. git-crypt - transparent file encryption in git. *GitHub*, November 2017a. https://github.com/AGWA/git-crypt, accessed 18 January 2018.

Ayer, A. git-crypt - transparent file encryption in git. *git-crypt Project Page*, 2017b. https://www.agwa.name/projects/git-crypt/, accessed 18 January 2018.

Bodden, E., Payer, M., and Athanasopoulos, E. Engineering Secure Software and Systems: 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings. Lecture Notes in Computer Science. Springer International Publishing, 2017. ISBN 9783319621050.
URL https://books.google.co.za/books?id=n3YpDwAAQBAJ

Bogdanovic, D., Sreenivasa, K., Huang, L., and Blair, D. Network access control list (acl) yang data model. *draft-ietf-netmod-acl-model-06. txt (work in progress)*, 2015.

Bradford, C. Why the cia triad is the new standard for information security. *Recovery Zone*, 2016. http://www.storagecraft.com/blog/why-the-cia-triad-is-the-new-standard-for-information-security/, accessed 18 October 2016.

Breithaupt, J. and Merkow, M. S. Information security principles of success. *IT Security Community Blog*, July 2014. http://www.pearsonitcertification.com/articles/article.aspx?p=2218577&seqNum=3, accessed 18 October 2016.

Buchanan, P. B. The fall of truecrypt and rise of veracrypt. *Medium*, November 2018. https://medium.com/asecuritysite-when-bob-met-alice/the-fall-of-truecrypt-and-rise-of-veracrypt-44f910ed5162, accessed 25 February 2019.

Chacon, S. Pro Git. Expert's voice in software development. Apress, 2009. ISBN 9781430218340.
URL https://books.google.co.za/books?id=HrTOa8-HPRYC

Chaudhuri, A., Ferrer, H., Prafullchandra, H., Sherry, J., Ng, K., Xiaoyu, G., Yao Sing, T., and Yiak Por, H. Best practices for mitigating risks in virtualized environments. *Cloud Security Alliance*, April 2015. https://downloads.cloudsecurityalliance.org/whitepapers/Best_Practices_for%20_Mitigating_Risks_Virtual_Environments_April2015_4-1-15_GLM5.pdf, accessed 15 July 2017.

Chess, B. and West, J. Secure Programming with Static Analysis. Addison-Wesley Software Security Series. Pearson Education, 2007. ISBN 9780132702027.
URL https://books.google.co.za/books?id=GL8AeTCu1WAC

Cisco. Cisco nac appliance. *Cisco*, 2014. https://www.cisco.com/c/en/us/products/security/nac-appliance-clean-access/index.html, accessed 2 April 2019.

Collins, W. cowboy. 2012.
URL https://www.dictionary.com/browse/cowboy

Commission, U. S. F. C. FCC Record: A Comprehensive Compilation of Decisions, Reports, Public Notices, and Other Documents of the Federal Communications Commission of the United States. The Commission, 2015.
URL https://books.google.co.za/books?id=TGdfrW_gcZMC

Cooper, M. Advanced Bash-Scripting Guide. Cooper, 2014.

Coos, A. Keeping source code safe with data loss prevention. *EndPoint Protector*, Data Loss Prevention, November 2018. https://www.endpointprotector.com/blog/keep-source-code-safe-with-dlp/.

Criado, J., Vicente-Chicote, C., Padilla, N., and Iribarne, L. A model-driven approach to graphical user interface runtime adaptation. In *Proceedings of the 5th Workshop on Models@run.time, Oslo, Norway, October 5th, 2010*, pages 49–59. 2010. URL http://ceur-ws.org/Vol-641/paper_15.pdf

Czerwonka, J. A. Method and system for supporting negative testing in combinatorial test case generators. August 9 2005. US Patent 6,928,393.

Davidoff, S. Data Breaches: Crisis and Opportunity. Pearson Education, 2019. ISBN 9780134507729.
URL https://books.google.co.za/books?id=QdaODwAAQBAJ

Death, D. Information Security Handbook: Develop a threat model and incident response strategy to build a strong information security framework. Packt Publishing, 2017. ISBN 9781788473262.
URL https://books.google.co.za/books?id=kMxPDwAAQBAJ

DeMarco, T. and Lister, T. Peopleware Productive Projects and Teams, Second Edition. Dorset House Publishing, New York, second edition edition, 1999.

Duclervil, S. R. and Liou, J.-C. The study of the effectiveness of the secure software development life-cycle models in it project management. In *16th International Conference on Information Technology-New Generations (ITNG 2019)*, pages 91–96. Springer, 2019.

Dwyer, I. Google's reference architecture. *ScaleFT Blog*, January 2017. https://www.scaleft.com/beyondcorp/, accessed 25 March 2019.

finide. How to monitor your system security with osquery on ubuntu 16.04. *Digital Ocean*, February 2017.
URL https://www.digitalocean.com/community/tutorials/how-to-monitor-your-system-security-with-osquery-on-ubuntu-16-04

Fitzgerald, T. Source code security the symantec way. *CustomerONE*, 2016. https://www.symantec.com/content/dam/symantec/docs/other-resources/customerone-source-code-security-the-symantec-way-en.pdf.

Graff, M. G. and van Wyk, K. R. Secure Coding: Principles and Practices. O'Reilly Media, 2003. ISBN 9780596002428.

Graziotin, D., Fagerholm, F., Wang, X., and Abrahamsson, P. What happens when software developers are (un)happy. *Journal of Systems and Software*, 140, 07 2017. doi:10.1016/j.jss.2018.02.041.

Gregg, J., Nam, M., Northcutt, S., and Pokladnik, M. Separation of duties in information technology. *SANS*, June 2017. https://www.sans.edu/cyber-research/security-laboratory/article/it-separation-duties, accessed 15 July 2017.

Grimes, R. A. How to restrict developers' admin rights. *CSO Online*, June 2012. http://www.csoonline.com/article/2617526/application-security/how-to-restrict-developers--admin-rights.html, accessed 15 July 2017.

Grimes, R. A. Developers must follow security rules, too. *CSO Online*, December 2014. http://www.csoonline.com/article/2862382/security/developers-must-follow-security-rules-too.html, accessed 15 July 2017.

Hardy, G. M. The critical security controls: What's NAC got to do with it? Technical report, SANS, April 2013. https://www.sans.org/reading-room/whitepapers/analyst/critical-security-controls-what-039-s-nac-it-35115, accessed 15 July 2017.

Heery, E. and Noon, M. knowledge worker. 2017.
URL https://www.oxfordreference.com/view/10.1093/acref/9780191827822.001.0001/acref-9780191827822-e-1688

Herbsleb, J. D. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, FOSE '07, pages 188–198. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0-7695-2829-5. doi:10.1109/FOSE.2007.11.
URL http://dx.doi.org/10.1109/FOSE.2007.11

Hernandez, S. Official (ISC) 2 guide to the CISSP CBK, Second Edition. CRC Press, second edition edition, 2012.

Hsu, T. Practical Security Automation and Testing: Tools and techniques for automated security scanning and testing in DevSecOps. Packt Publishing, 2019. ISBN 9781789611694.
URL https://books.google.co.za/books?id=z4KGDwAAQBAJ

Khan, M. E., Khan, F. *et al.* A comparative study of white box, black box and grey box testing techniques. *Int. J. Adv. Comput. Sci. Appl*, 3(6), 2012.

Kissel, R. L., Stine, K. M., Scholl, M. A., Rossman, H., Fahlsing, J., and Gulick, J. Security considerations in the system development life cycle. *Special Publication (NIST SP)-800-64 Rev 2*, 2008.

Lane, N., Conklin, W., White, G., and Williams, D. CASP+ CompTIA Advanced Security Practitioner Certification All-in-One Exam Guide, Second Edition (Exam CAS-003). McGraw-Hill Education, 2019. ISBN 9781260441345.
URL https://books.google.co.za/books?id=klWQDwAAQBAJ

Lee, J., Woo, J., Lee, C., and Joo, K. A software development methodology for secure web application. *International Journal on Advanced Science, Engineering and Information Technology*, 9(1):336–341, 2019.

Lee, Y.-J., Lim, J.-S., Ji, J.-H., Cho, H.-G., and Woo, G. Plagiarism detection among source codes using adaptive methods. *KSII Transactions on Internet & Information Systems*, 6(6), 2012.

Loeliger, J. and McCullough, M. Version Control with Git: Powerful tools and techniques for collaborative software development. O'Reilly Media, 2012. ISBN 9781449345044.
URL https://books.google.co.za/books?id=qIucp61eqAwC

McCance, G. Sdcd12: Supporting science with cloud computing. 2012.
URL https://www.slideshare.net/gmccance/cern-data-centre-evolution

Metz, C. How github conquered google, microsoft, and everyone else. *Wired. com*, pages 676–689, 2015.

Meucci, M. and Muller, A. OWASP Testing Guide 4.0. Testing Guide. OWASP, 2013.

Meyer, A., Barr, E. T., Bird, C., and Zimmermann, T. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering*, 2019.

Microsoft. Network policy and access services overview. *TechNET Documents*, 2016. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/hh831683(v=ws.11), accessed 3 April 2019.

Mogull, R. Understanding and selecting a data loss prevention solution. *SANS Institute*, n.d.

URL https://securosis.com/assets/library/publications/DLP-Whitepaper.pdf

Nagy, Z. Soft Skills to Advance Your Developer Career: Actionable Steps to Help Maximize Your Potential. Apress, 2019. ISBN 9781484250914.
URL https://books.google.co.za/books?id=Sm2fxgEACAAJ

Novak, M., Joy, M., and Kermek, D. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)*, 19(3):1–37, 2019.

Ozgen. How to block source codes. *My DLP*, 2018. https://www.mydlp.com/block-source-codes/.

Pandey, K. L., Agarwal, S., Misra, S., and Prasad, R. Plagiarism detection in software using efficient string matching. In **Murgante, B., Gervasi, O., Misra, S., Nedjah, N., Rocha, A. M. A. C., Taniar, D., and Apduhan, B. O.**, editors, *Computational Science and Its Applications – ICCSA 2012*, pages 147–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-31128-4.

Pruteanu, A. Becoming the Hacker: The Playbook for Getting Inside the Mind of the Attacker. Packt Publishing, 2019. ISBN 9781788623759.
URL https://books.google.co.za/books?id=hyOGDwAAQBAJ

Qian, J., Hinrichs, S., and Nahrstedt, K. ACLA: A Framework for Access Control List (ACL) Analysis and Optimization, pages 197–211. Springer US, Boston, MA, 2001. ISBN 978-0-387-35413-2. doi:10.1007/978-0-387-35413-2_18.
URL https://doi.org/10.1007/978-0-387-35413-2_18

Rhodes-Ousley, M. Information Security - The Complete Reference, Second Edition. McGraw-Hill, second edition edition, 2013.

Rota, G. Storing sensitive data in a git repository using git-crypt. *TwinBit*, March 2014. http://www.twinbit.it/en/blog/storing-sensitive-data-git-repository-using-git-crypt, accessed 18 January 2018.

Runeson, P. A survey of unit testing practices. *IEEE software*, 23(4):22–29, 2006.

Salim, N. Engaged software developers can result in happy clients. *Fringent*, March 2017. https://www.fingent.com/blog/engaged-software-developers-can-result-in-happy-clients, accessed 15 July 2017.

Sandhu, R. S. and Samarati, P. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

Santacroce, F., Olsson, A., Voss, R., and Narebski, J. Git: Mastering Version Control. Packt Publishing, 2016. ISBN 9781787123205.

Schedl, T. Ntfs security model. *Coopware*, 2019. http://www.coopware.in2.info/_ntfsacl_ht.htm, accessed 8 October 2019.

Shahnaz, F. A study of human factors in interactive software & designing of an efficient user interface tool. Ph.D. thesis, Integral University, 2013.

Shang, N. Transparent git encryption. *GitHub Gist*, March 2011. https://gist.github.com/shadowhand/873637, accessed 18 January 2018.

Shapland, R. Introduction to network access control products in the enterprise. *Techtarget*, 2015. http://searchsecurity.techtarget.com/feature/Introduction-to-network-access-control-products-in-the-enterprise, accessed 15 July 2017.

Shinde, V. A complete non-functional testing guide for beginners. *Software Testing Help*, September 2019. https://www.softwaretestinghelp.com/what-is-non-functional-testing/, accessed 11 October 2019.

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., and Brechmann, A. Understanding understanding source code with functional magnetic resonance imaging. *ACM*, 05 2014. doi:10.1145/2568225.2568252.

Singh, B. and Gautam, S. The impact of software development process on software quality: A review. In *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 666–672. 12 2016. doi:10.1109/CICN.2016.137.

Tang, P. and Liu, H.-Y. Security enabled network access control. February 27 2007. US Patent 7,185,365.

Vasbinder, E. How to make the most of access control lists. *Computerworld*, November 2003. http://www.computerworld.com/article/2573380/security0/how-to-make-the-most-of-access-control-lists.html, accessed 16 July 2017.

Verdon, D. Security policies and the software developer. *IEEE security & privacy*, 4(4):42–49, 2006.

Warren, T. Microsoft confirms some windows 10 source code has leaked. *The Verge*, June 2017. https://www.theverge.com/2017/6/24/15867350/microsoft-windows-10-source-code-leak, accessed 15 July 2017.

Westby, E. Git for Teams: A User-Centered Approach to Creating Efficient Workflows in Git. O'Reilly Media, 2015. ISBN 9781491911228.
URL https://books.google.co.za/books?id=ynFrCgAAQBAJ

Whitman, M. and Mattord, H. Principles of Information Security. Cengage Learning, 2014. ISBN 9781285448367.
URL https://books.google.co.za/books?id=htwbCgAAQBAJ

Whitton, S. Storing sensitive data in a git repository using git-crypt. *spwhitton.name*, December 2016. https://spwhitton.name/tech/code/git-remote-gcrypt/, accessed 18 January 2018.

Wienclaw, R. A. Quantitative and qualitative analysis. *Salem Press Encyclopedia*, 2019.
URL http://0-search.ebscohost.com.wam.seals.ac.za/login.aspx?direct=true&db=ers&AN=89185655&site=eds-live

Williams, J. Code review introduction. *OWASP*, September 2010. https://www.owasp.org/index.php/Code_Review_Introduction, accessed 15 July 2017.

Xiang, S. and Zhu, Z. Dynamic access control of encrypted data in cloud computing environment. *International Journal of Performability Engineering*, 15(3), 2019.

zimbatm. After 4 months of usage, i don't recommend using git-crypt. *Hacker News*, April 2014. https://news.ycombinator.com/item?id=7508734, accessed 18 January 2018.

Appendix A

Rhodes Ethics Documentation

RHODES UNIVERSITY
Where leaders learn

Human Ethics subcommittee
Rhodes University Ethical Standards Committee
PO Box 94, Grahamstown, 6140, South Africa
t: +27 (0) 46 603 8055
f: +27 (0) 46 603 8822
e: ethics-committee@ru.ac.za

www.ru.ac.za/research/research/ethics
NHREC Registration no. REC-241114-045

29 July 2019

Grant Ongers

Review Reference: 2019-0611-700

Email: g16o7515@campus.ru.ac.za

Dear Grant Ongers

TITLE: Securing Software Development using Developer Access Control,

Principal Investigator: Dr. Yusuf Motara

Collaborators: Mr. Grant Ongers,

This letter confirms that the above research proposal has been reviewed and **APPROVED** by the Rhodes University Ethical Standards Committee (RUESC) – Human Ethics (HE) sub-committee.

Approval has been granted for 1 year. An annual progress report will be required in order to renew approval for an additional period. You will receive an email notifying when the annual report is due.

Please ensure that the ethical standards committee is notified should any substantive change(s) be made, for whatever reason, during the research process. This includes changes in investigators. Please also ensure that a brief report is submitted to the ethics committee on completion of the research. The purpose of this report is to indicate whether the research was conducted successfully, if any aspects could not be completed, or if any problems arose that the ethical standards committee should be aware of. If a thesis or dissertation arising from this research is submitted to the library's electronic theses and dissertations (ETD) repository, please notify the committee of the date of submission and/or any reference or cataloguing number allocated.

Sincerely

Prof Joanna Dames

Chair: Human Ethics sub-committee, RUESC- HE

Appendix B

Code Corral Source

Listing B.1: Code Corral / code-corral

```
#!/bin/bash
set -e

CODE_CORRAL_CLIENT_ARGS=$@

CODE_CORRAL_CONFIGURATION_FOLDER="$HOME/.code-corral"
CODE_CORRAL_CONFIGURATION_FILE="${CODE_CORRAL_CONFIGURATION_FOLDER}/config"

CODE_CORRAL_SERVER="https://code-corral.herokuapp.com"

#
    ------------------------------------------------------------------------

# Set up logs and logging
#
    ------------------------------------------------------------------------


if [ -z $LOG_LEVEL ]
then
  LOG_LEVEL='info'
else
  LOG_LEVEL=$(echo $LOG_LEVEL|tr '[:upper:]' '[:lower:]')
fi

function log {
  log_level=$1
  shift
  message=$@
  case $log_level in
    debug)
      if [ $LOG_LEVEL == 'debug' ]
      then
        echo -e "[DEBUG] $message" 1>&2
      fi
      ;;
    info)
      if [ $LOG_LEVEL == 'debug' ] || [ $LOG_LEVEL == 'info' ]
      then
        echo -e "[INFO] $message" 1>&2
      fi
      ;;
    warning)
```

```
40        if [ $LOG_LEVEL != 'error' ]
41        then
42          echo -e "[WARNING] $message" 1>&2
43        fi
44        ;;
45      error)
46        echo -e "[ERROR] $message" 1>&2
47        ;;
48      *)
49        log error "Unknown log level \"$log_level\" with message: $message"
50        ;;
51    esac
52  }
53
54  #---------------------------------------------------------------------------
55  # Code Corral configuration
56  #---------------------------------------------------------------------------
57
58  mkdir -p "${CODE_CORRAL_CONFIGURATION_FOLDER}"
59
60  if [ ! -f "${CODE_CORRAL_CONFIGURATION_FILE}" ]
61  then
62    touch "${CODE_CORRAL_CONFIGURATION_FILE}"
63  fi
64
65  source "${CODE_CORRAL_CONFIGURATION_FILE}"
66
67  if [ -z "${CODE_CORRAL_EMAIL_ADDRESS}" ]
68  then
69    log error "No CODE_CORRAL_EMAIL_ADDRESS defined in ${
       CODE_CORRAL_CONFIGURATION_FILE}"
70    exit -1
71  fi
72
73  #---------------------------------------------------------------------------
74  # Non-configurable or generated configuration
75  #---------------------------------------------------------------------------
```

```
76
77  CODE_CORRAL_API_VERSION="1.1"
78
79  CODE_CORRAL_BINARY_LOCATION=/opt/code-corral/bin
80  CODE_CORRAL_BINARY_NAME=code-corral
81  CODE_CORRAL_BINARY=$CODE_CORRAL_BINARY_LOCATION/${CODE_CORRAL_BINARY_NAME}
82  VERACRYPT_BINARY=$CODE_CORRAL_BINARY_LOCATION/veracrypt
83  OSQUERY_BINARY=$CODE_CORRAL_BINARY_LOCATION/osqueryi
84  CODE_CORRAL_URI="${CODE_CORRAL_SERVER}/api/${CODE_CORRAL_API_VERSION}"
85
86  #
        ------------------------------------------------------------------------
87  # Helper functions
88  #
        ------------------------------------------------------------------------
89
90  function ensure_curl_is_installed {
91    set +e
92    CURL=$(which curl)
93    set -e
94    if [ -z $CURL ]
95    then
96      log warning "Curl is a required dependency. Installing ..." 1>&2
97      sudo apt-get install curl 1>/dev/null
98    fi
99  }
100
101 function ensure_installation_folders {
102   sudo mkdir -p /opt/code-corral/bin
103   sudo chmod 755 /opt/code-corral
104   sudo chmod 755 /opt/code-corral/bin
105 }
106
107 function respawn {
108   log warning "Automatically re-executing $CODE_CORRAL_BINARY
        $CODE_CORRAL_CLIENT_ARGS\n"
109   $CODE_CORRAL_BINARY $CODE_CORRAL_CLIENT_ARGS
110   exit $?
111 }
112
113 function download_install_and_execute_code_corral_client {
```

```
114   ensure_installation_folders
115   error_log=`mktemp`
116   tmp_binary=`mktemp`
117   full_url="$CODE_CORRAL_SERVER/client/bash/${CODE_CORRAL_BINARY_NAME}"
118   set +e
119   curl --fail "${full_url}" 1>"$tmp_binary" 2>"${error_log}"
120   curl_status_code=$?
121   set -e
122   if [[ $curl_status_code != 0 ]]
123   then
124     log error "Download of ${full_url} failed with status: ${
      curl_status_code}" 1>&2
125     cat "${error_log}"
126     exit ${curl_status_code}
127   fi
128   sudo mv "$tmp_binary" "$CODE_CORRAL_BINARY"
129   sudo chmod 755 $CODE_CORRAL_BINARY
130   log info "Installation complete, please add $CODE_CORRAL_BINARY_LOCATION
      to your PATH environment:"
131   log info "   export PATH=$CODE_CORRAL_BINARY_LOCATION:\$PATH"
132   log info " after which you will be able to use code-corral"
133   respawn
134 }
135
136 function download_and_install_veracrypt {
137   ensure_installation_folders
138   error_log=`mktemp`
139   tmp_binary=`mktemp`
140   if [[ "$OSTYPE" == "darwin" ]]
141   then
142     os_type="osx"
143   elif [[ "$OSTYPE" == "msys" ]]
144   then
145     os_type="win"
146   else
147     os_type="linux"
148   fi
149   full_url="$CODE_CORRAL_SERVER/veracrypt/$os_type/veracrypt"
150   set +e
151   curl --fail "${full_url}" 1>"$tmp_binary" 2>"${error_log}"
152   curl_status_code=$?
153   if [[ $curl_status_code != 0 ]]
154   then
155     log error "Download of ${full_url} failed with status: ${
```

```
    curl_status_code}"
156     cat "${error_log}"
157     exit ${curl_status_code}
158   fi
159   set -e
160   sudo mv "$tmp_binary" "$VERACRYPT_BINARY"
161   sudo chmod 755 "$VERACRYPT_BINARY"
162 }
163
164 function download_and_install_osquery {
165   ensure_installation_folders
166   error_log=`mktemp`
167   tmp_binary=`mktemp`
168   if [[ "$OSTYPE" == "darwin" ]]
169   then
170     os_type="osx"
171   elif [[ "$OSTYPE" == "msys" ]]
172   then
173     os_type="win"
174   else
175     os_type="linux"
176   fi
177   full_url="$CODE_CORRAL_SERVER/osquery/$os_type/osqueryi"
178   set +e
179   curl --fail "${full_url}" 1>"$tmp_binary" 2>"${error_log}"
180   curl_status_code=$?
181   if [[ $curl_status_code != 0 ]]
182   then
183     log error "Download of ${full_url} failed with status: ${
    curl_status_code}"
184     cat "${error_log}"
185     exit ${curl_status_code}
186   fi
187   set -e
188   sudo mv "$tmp_binary" "$OSQUERY_BINARY"
189   sudo chmod 755 "$OSQUERY_BINARY"
190 }
191
192 function download_and_install_client {
193   ensure_installation_folders
194   error_log=`mktemp`
195   tmp_binary=`mktemp`
196   full_url="$CODE_CORRAL_SERVER/client/bash/${CODE_CORRAL_BINARY_NAME}"
197   set +e
```

```
198   curl --fail "${full_url}" 1>"$tmp_binary" 2>"${error_log}"
199   curl_status_code=$?
200   if [[ $curl_status_code != 0 ]]
201   then
202     log error "Download of ${full_url} failed with status: ${
      curl_status_code}" 1>&2
203     cat "${error_log}"
204     exit ${curl_status_code}
205   fi
206   set -e
207   sudo mv "$tmp_binary" "$CODE_CORRAL_BINARY"
208   sudo chmod 755 "$CODE_CORRAL_BINARY"
209   respawn
210 }
211
212 function ensure_veracrypt_is_installed {
213   if [ ! -f $VERACRYPT_BINARY ]
214   then
215     log warning "Veracrypt is a required dependency. Installing ..." 1>&2
216     download_and_install_veracrypt
217   fi
218 }
219
220 function ensure_osquery_is_installed {
221   if [ ! -f $OSQUERY_BINARY ]
222   then
223     log warning "OSQuery is a required dependency. Installing ..." 1>&2
224     download_and_install_osquery
225   fi
226 }
227
228 function close_project {
229   project_name=$1
230
231   if [ -z $project_name ]
232   then
233     log error "No project name provided!"
234     exit -1
235   fi
236
237   ${VERACRYPT_BINARY} -d "${CODE_CORRAL_CONFIGURATION_FOLDER}/${
      project_name}.hc"
238 }
239
```

```
240 function release_project {
241   project_name=$1
242   clone_path=$2
243
244   if [ -z $project_name ]
245   then
246     log error "No project name provided!"
247     exit -1
248   fi
249
250   ensure_project_not_already_mounted "$project_name"
251
252   rm -f "${CODE_CORRAL_CONFIGURATION_FOLDER}/${project_name}."*
253
254   api_post "project/release/${project_name}"
255 }
256
257 function ensure_project_not_already_mounted {
258   set +e
259   "${VERACRYPT_BINARY}" -l "${CODE_CORRAL_CONFIGURATION_FOLDER}/$1.hc" 2>/
        dev/null
260   if [ $? -eq 0 ]
261   then
262     log error "Project $1 is currently mounted"
263     exit -1
264   fi
265   set -e
266 }
267
268 function open_project {
269   project_name=$1
270   clone_path=$2
271
272   if [ -z $project_name ]
273   then
274     log error "No project name provided!"
275     exit -1
276   fi
277
278   ensure_project_not_already_mounted "$project_name"
279
280   project_config_path="${CODE_CORRAL_CONFIGURATION_FOLDER}/${project_name}.
        config"
281   if [ ! -f "${project_config_path}" ]
```

```
282   then
283     log error "No configuration file found for this project"
284     exit -1
285   fi
286
287   if [ -z $clone_path ]
288   then
289     clone_path=$(cat ${project_config_path})
290   fi
291
292   mkdir -p "${clone_path}"
293   clone_path="$(cd "$(dirname "$clone_path")" && pwd)/$(basename "
      $clone_path")"
294
295   api_post "project/open/${project_name}"
296 }
297
298 function clone_project {
299   project_name=$1
300   clone_path=$2
301
302   if [ -z $project_name ]
303   then
304     log error "No project name provided!"
305     exit -1
306   fi
307
308   ensure_project_not_already_mounted "$project_name"
309
310   if [ -z $clone_path ]
311   then
312     log error "No project clone path provided!"
313     exit -1
314   fi
315
316   project_config_path="${CODE_CORRAL_CONFIGURATION_FOLDER}/${project_name}.
      config"
317   if [ -f "${project_config_path}" ]
318   then
319     log error "This project already has a local configuration path, and
      cannot be re-cloned"
320     exit -1
321   fi
322
```

```
323   if [ -f "${clone_path}" ]
324   then
325     log error "Cannot create path ${clone_path}"
326     exit -1
327   fi
328
329   mkdir -p "${clone_path}"
330   clone_path="$(cd "$(dirname "$2")" && pwd)/$(basename "$2")"
331
332   api_post "project/clone/${project_name}"
333
334   echo "${clone_path}" > "${project_config_path}"
335 }
336
337 function handle_api_result {
338   log debug "API Result: $@"
339   IFS='|' read -ra api_commands <<< "$@"
340   case ${api_commands[0]} in
341     pong)
342       # no need to do anything here
343       ;;
344     update)
345       case ${api_commands[1]} in
346         client)
347           log warning "Code Corral Client is out of date, downloading and
    installing new client" 1>&2
348           download_and_install_client
349           ;;
350         veracrypt)
351           log warning "Veracrypt is out of date, downloading and installing
     update" 1>&2
352           download_and_install_veracrypt
353           respawn
354           ;;
355         osquery)
356           log warning "OSQuery is out of date, downloading and installing
    update" 1>&2
357           download_and_install_osquery
358           respawn
359           ;;
360         *)
361           log error "Unhandled Update Type: $@"
362           exit -1
363         ;;
```

```
364       esac
365       ;;
366     osquery)
367       case ${api_commands[1]} in
368         q)
369           log info "Answering remote host query with ID:${api_commands[2]} about local system"
370           data=$(echo "${api_commands[3]}"|${OSQUERY_BINARY} --csv|tail -n 1)
371           api_post "osquery/answer/${api_commands[2]}" $data
372           ;;
373         ack)
374           respawn
375           ;;
376         *)
377           log error "Unhandled OSQuery Type: $@"
378           exit -1
379           ;;
380       esac
381       ;;
382     project)
383       case ${api_commands[1]} in
384         clone)
385           log info "Cloning project ${api_commands[2]} into ${clone_path}"
386           ensure_ssh_auth_headers_are_loaded
387           encryption_key=$(echo "${CODE_CORRAL_EMAIL_ADDRESS}|${SSH_FINGERPRINT}|${api_commands[5]}"|${SHA_SUM}|cut -f1 -d' ')
388           log info "Creating new encrypted volume for ${api_commands[2]}"
389           if [[ "$OSTYPE" == "darwin" ]]
390           then
391             fs_type="fat"
392             rand=""
393           elif [[ "$OSTYPE" == "msys" ]]
394           then
395             fs_type="fat"
396             rand=""
397           else
398             fs_type="ext3"
399             rand="--random-source=/dev/urandom"
400           fi
401           ${VERACRYPT_BINARY} --create --volume-type=normal --size=${api_commands[4]}M --encryption=AES --hash=SHA-256 --filesystem=${fs_type} --password="${encryption_key}" --pim=1 --keyfiles='' ${rand} "${CODE_CORRAL_CONFIGURATION_FOLDER}/${api_commands[2]}.hc"
```

```
402           log info "Formatting new encrypted volume for ${api_commands[2]}
    to ${fs_type}"
403           log info "Mounting new encrypted volume for ${api_commands[2]} at
     ${clone_path}"
404           ${VERACRYPT_BINARY} --password="${encryption_key}" --pim=1 --
    keyfiles='' --protect-hidden=no --filesystem=ext3 "${
    CODE_CORRAL_CONFIGURATION_FOLDER}/${api_commands[2]}.hc" "${clone_path}"
405           cd "${clone_path}"
406           sudo mkdir "${api_commands[2]}"
407           sudo chown `whoami`:`whoami` "${api_commands[2]}"
408           cd "${api_commands[2]}"
409           git init
410           git remote add origin "${api_commands[3]}"
411           git remote update
412           git checkout -b master origin/master >/dev/null
413           ;;
414         released)
415           log info "Project ${api_commands[2]} has been released"
416           ;;
417         open)
418           ensure_ssh_auth_headers_are_loaded
419           old_encryption_key=$(echo "${CODE_CORRAL_EMAIL_ADDRESS}|${
    SSH_FINGERPRINT}|${api_commands[3]}"|${SHA_SUM}|cut -f1 -d' ')
420           new_encryption_key=$(echo "${CODE_CORRAL_EMAIL_ADDRESS}|${
    SSH_FINGERPRINT}|${api_commands[4]}"|${SHA_SUM}|cut -f1 -d' ')
421           log info "Recrypting project ${api_commands[2]}"
422           ${VERACRYPT_BINARY} --change --pim=1 --keyfiles='' --password="${
    old_encryption_key}" --new-pim=1 --new-keyfiles='' --new-password="${
    new_encryption_key}" --random-source=/dev/urandom "${
    CODE_CORRAL_CONFIGURATION_FOLDER}/${api_commands[2]}.hc" 1>/dev/null
423           log info "Opening project ${api_commands[2]} into ${clone_path}"
424           ${VERACRYPT_BINARY} --password="${new_encryption_key}" --pim=1 --
    keyfiles='' --protect-hidden=no --filesystem=ext3 "${
    CODE_CORRAL_CONFIGURATION_FOLDER}/${api_commands[2]}.hc" "${clone_path}"
425           cd "${clone_path}/${api_commands[2]}"
426           ;;
427         list)
428           echo "The list of projects you have access to:"
429           for project_name in ${api_commands[@]:2}
430           do
431             project_config_path="${CODE_CORRAL_CONFIGURATION_FOLDER}/${
    project_name}.config"
432             echo -n "  * ${project_name}"
433             if [ -f "${project_config_path}" ]
```

```
434             then
435               project_working_directory=$(cat "$project_config_path")
436               echo -n " [${project_working_directory}]"
437             fi
438             echo ''
439           done
440           ;;
441         *)
442           log error "Unhandled Project Type: $@"
443           exit -1
444         ;;
445       esac
446       ;;
447     *)
448       log error "Unhandled API Result: $@"
449       exit -1
450   esac
451 }
452
453 function api_get {
454   make_api_call 'GET' $@
455 }
456
457 function api_post {
458   make_api_call 'POST' $@
459 }
460
461 function ensure_ssh_auth_headers_are_loaded {
462   if [ -z $USER_AUTH_HEADER ]
463   then
464     while read ssh_line
465     do
466       possible_ssh_keyfile=$(echo ${ssh_line}|cut -f3 -d' ')
467       log debug "Attempting to add ssh key file: ${possible_ssh_keyfile}"
468       if [ -f "$possible_ssh_keyfile" ]
469       then
470         if [ -z "$SSH_KEYFILE" ]
471         then
472           export SSH_KEYFILE="$possible_ssh_keyfile"
473           export SSH_FINGERPRINT=$(echo ${ssh_line}|cut -f2 -d' '|cut -d: -
    f2)
474         else
475           log warning "Multiple ssh key files detected, using ${SSH_KEYFILE
    }, skipping ${possible_ssh_keyfile}"
```

```
476         fi
477       fi
478     done < <(ssh-add -l)
479
480     if [ -z "$SSH_KEYFILE" ]
481     then
482       log error "Unable to find any loaded ssh keys. Have you added any
    with \'ssh-add\'?"
483       exit -1
484     fi
485
486     log info "Using ssh key file: ${SSH_KEYFILE}"
487
488     export USER_AUTH_HEADER=$(echo -n "${CODE_CORRAL_EMAIL_ADDRESS}|${
    CLIENT_SHA}"|openssl pkeyutl -sign -inkey "${SSH_KEYFILE}"|base64 --wrap
     0)
489     if [ -z "$USER_AUTH_HEADER" ]
490     then
491       log error "Failed encryption test using ssh keyfile: ${SSH_KEYFILE}"
492       exit -1
493     fi
494   fi
495 }
496
497 function make_api_call {
498   request_method="$1"
499   api_call="$2"
500   data="$3"
501
502   ensure_ssh_auth_headers_are_loaded
503
504   # Execute API call
505   full_url="$CODE_CORRAL_URI/${api_call}"
506   log debug "Sending API Call : ${full_url}"
507   error_log=`mktemp`
508   curl_output=`mktemp`
509   http_status_code=$(curl --request "${request_method}" --silent --output $
    {curl_output} -w '%{http_code}' \
510     --header "X-Code-Corral-Email-Address: ${CODE_CORRAL_EMAIL_ADDRESS}" \
511     --header "X-Code-Corral-Client: ${CLIENT_SHA}" \
512     --header "X-Code-Corral-Tool-Veracrypt: ${VERACRYPT_SHA}" \
513     --header "X-Code-Corral-Tool-OSQuery: ${OSQUERY_SHA}" \
514     --header "X-Code-Corral-Auth: ${USER_AUTH_HEADER}" \
515     --data "${data}" \
```

```
516     --stderr "${error_log}" "${full_url}")
517   api_result=$(cat "${curl_output}")
518   if [[ $http_status_code -ne 200 ]]
519   then
520     log error "Failed upstream API call ${full_url} with status:
      $http_status_code $api_result" 1>&2
521     cat "${error_log}"
522     exit -1
523   fi
524   handle_api_result $api_result
525 }
526
527 function show_help {
528   echo "$CODE_CORRAL_BINARY_NAME help                                  -
      shows this command-line help"
529   echo "$CODE_CORRAL_BINARY_NAME project list                          -
      lists all projects you have access to"
530   echo "$CODE_CORRAL_BINARY_NAME project clone <project_name> <path> -
      clone the named project into the destination path"
531   echo "$CODE_CORRAL_BINARY_NAME project release <project_name>        -
      release local project and remove from drive, it can now be cloned
      elsewhere"
532   echo "$CODE_CORRAL_BINARY_NAME project open <project_name> [path]  -
      decrypt and open project to continue work, optionally at a different
      path"
533   echo "$CODE_CORRAL_BINARY_NAME project close <project_name>         -
      close (unmount) project"
534 }
535
536 #
      ------------------------------------------------------------------------
537 # Dependencies are up-to-date and correct
538 #
      ------------------------------------------------------------------------
539
540 if [ "$0" != "$CODE_CORRAL_BINARY" ]
541 then
542   log warning "Invalid client found, downloading new client"
543   download_install_and_execute_code_corral_client
544 fi
545
546 ensure_curl_is_installed
```

```
547 ensure_veracrypt_is_installed
548 ensure_osquery_is_installed
549
550 #
    ------------------------------------------------------------------------
551 # Create client version hashes for API calls
552 #
    ------------------------------------------------------------------------
553
554 SHA_SUM_EXECUTABLE=shasum
555 SHA_SUM_BINARY=`which ${SHA_SUM_EXECUTABLE}`
556 if [ -z ${SHA_SUM_BINARY} ]
557 then
558   log error "Unfortunately, ${SHA_SUM_EXECUTABLE} is required!"
559   exit -1
560 fi
561 SHA_SUM="${SHA_SUM_BINARY} -a 256"
562
563 CLIENT_SHA=$(${SHA_SUM} < "$CODE_CORRAL_BINARY" | cut -f1 -d' ')
564 VERACRYPT_SHA=$(${SHA_SUM} < "$VERACRYPT_BINARY" | cut -f1 -d' ')
565 OSQUERY_SHA=$(${SHA_SUM} < "$OSQUERY_BINARY" | cut -f1 -d' ')
566
567 #
    ------------------------------------------------------------------------
568 # Redirect process flow according to command-line arguments
569 #
    ------------------------------------------------------------------------
570
571 if [ -z "$1" ]
572 then
573   show_help
574   exit -1
575 fi
576
577 call_command=$1
578 shift
579
580 case $call_command in
581   help)
582     show_help
```

```
583     ;;
584   project)
585     sub_command=$1
586     if [ -z "${sub_command}" ]
587     then
588       log error "The project command requires a sub command"
589       show_help
590       exit -1
591     fi
592     shift
593     case $sub_command in
594       clone)
595         clone_project $@
596         ;;
597       open)
598         open_project $@
599         ;;
600       release)
601         release_project $@
602         ;;
603       close)
604         close_project $@
605         ;;
606       list)
607         api_get "project/list"
608         ;;
609       *)
610         log error "Unhandled project sub-command: $sub_command"
611         exit -1
612         ;;
613     esac
614     ;;
615   *)
616     log error "Unhandled command-line arguments: $@"
617     show_help
618     exit -1
619     ;;
620 esac
621
622 exit 0
```

Listing B.2: Code Corral / Error Messages

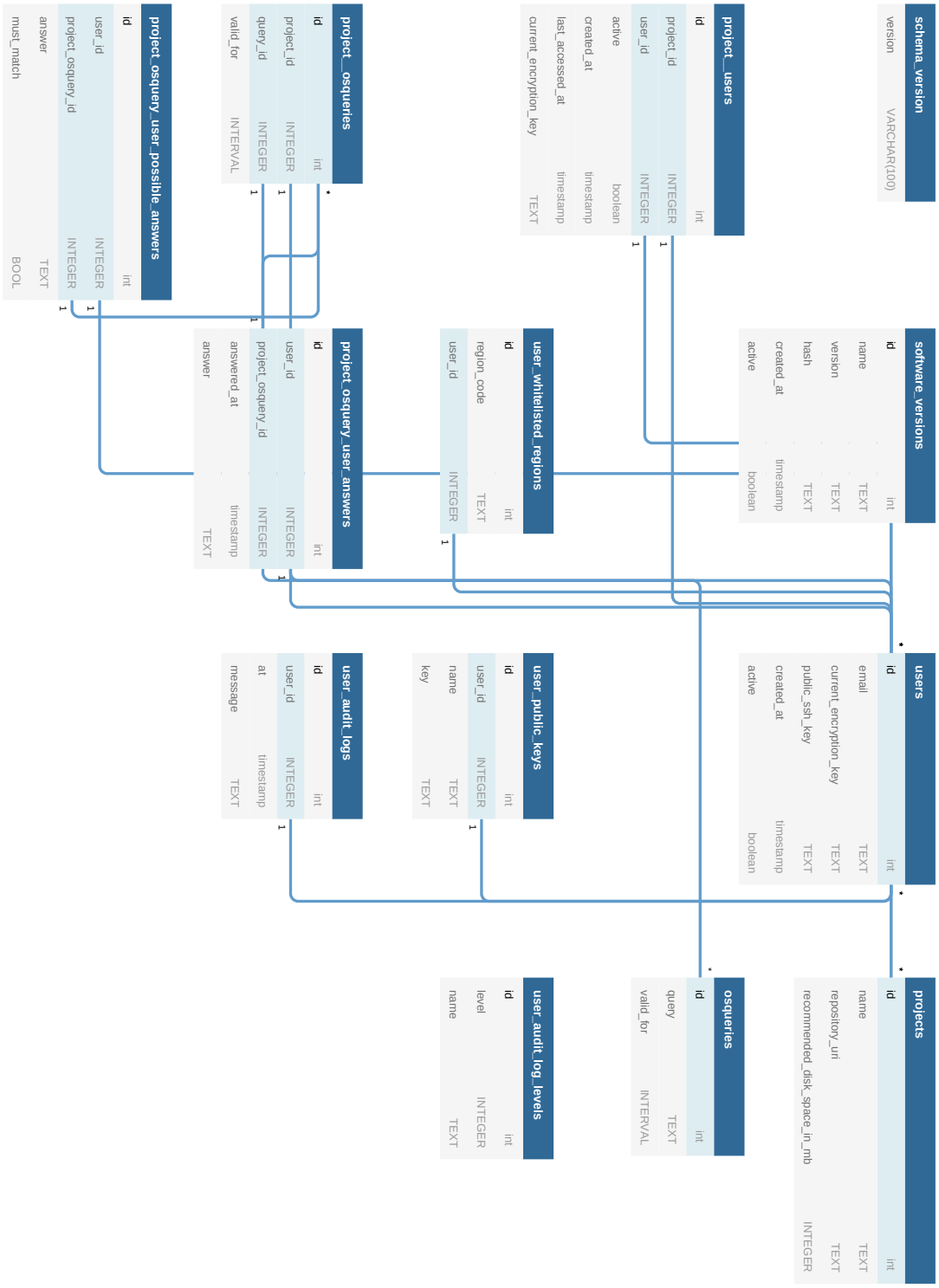
```
1       log error "Unknown log level \"$log_level\" with message: $message"
2   log error "No CODE_CORRAL_EMAIL_ADDRESS defined in ${
    CODE_CORRAL_CONFIGURATION_FILE}"
3     log error "Download of ${full_url} failed with status: ${
    curl_status_code}" 1>&2
4     log error "Download of ${full_url} failed with status: ${
    curl_status_code}"
5     log error "Download of ${full_url} failed with status: ${
    curl_status_code}"
6     log error "Download of ${full_url} failed with status: ${
    curl_status_code}" 1>&2
7     log error "No project name provided!"
8     log error "No project name provided!"
9     log error "Project $1 is currently mounted"
10    log error "No project name provided!"
11    log error "No configuration file found for this project"
12    log error "No project name provided!"
13    log error "No project clone path provided!"
14    log error "This project already has a local configuration path, and
    cannot be re-cloned"
15    log error "Cannot create path ${clone_path}"
16          log error "Unhandled Update Type: $@"
17          log error "Unhandled OSQuery Type: $@"
18          log error "Unhandled Project Type: $@"
19      log error "Unhandled API Result: $@"
20      log error "Unable to find any loaded ssh keys. Have you added any
    with \`ssh-add\`?"
21      log error "Failed encryption test using ssh keyfile: ${SSH_KEYFILE}"
22    log error "Failed upstream API call ${full_url} with status:
    $http_status_code $api_result" 1>&2
23  log error "Unfortunately, ${SHA_SUM_EXECUTABLE} is required!"
24      log error "The project command requires a sub command"
25        log error "Unhandled project sub-command: $sub_command"
26    log error "Unhandled command-line arguments: $@"
```

Listing B.3: Code Corral / Warning Messages

```
1     log warning "Curl is a required dependency. Installing ..." 1>&2
2   log warning "Automatically re-executing $CODE_CORRAL_BINARY
     $CODE_CORRAL_CLIENT_ARGS\n"
3     log warning "Veracrypt is a required dependency. Installing ..." 1>&2
4     log warning "OSQuery is a required dependency. Installing ..." 1>&2
5           log warning "Code Corral Client is out of date, downloading and
     installing new client" 1>&2
6           log warning "Veracrypt is out of date, downloading and installing
      update" 1>&2
7           log warning "OSQuery is out of date, downloading and installing
     update" 1>&2
8           log warning "Multiple ssh key files detected, using ${SSH_KEYFILE
     }, skipping ${possible_ssh_keyfile}"
9   log warning "Invalid client found, downloading new client"
```

Listing B.4: Code Corral / Info Messages

```
1       log error "Unknown log level \"$log_level\" with message: $message"
2   log error "No CODE_CORRAL_EMAIL_ADDRESS defined in ${
    CODE_CORRAL_CONFIGURATION_FILE}"
3     log error "Download of ${full_url} failed with status: ${
    curl_status_code}" 1>&2
4     log error "Download of ${full_url} failed with status: ${
    curl_status_code}"
5     log error "Download of ${full_url} failed with status: ${
    curl_status_code}"
6     log error "Download of ${full_url} failed with status: ${
    curl_status_code}" 1>&2
7     log error "No project name provided!"
8     log error "No project name provided!"
9     log error "Project $1 is currently mounted"
10    log error "No project name provided!"
11    log error "No configuration file found for this project"
12    log error "No project name provided!"
13    log error "No project clone path provided!"
14    log error "This project already has a local configuration path, and
    cannot be re-cloned"
15    log error "Cannot create path ${clone_path}"
16          log error "Unhandled Update Type: $@"
17          log error "Unhandled OSQuery Type: $@"
18          log error "Unhandled Project Type: $@"
19      log error "Unhandled API Result: $@"
20      log error "Unable to find any loaded ssh keys. Have you added any
    with \`ssh-add\`?"
21      log error "Failed encryption test using ssh keyfile: ${SSH_KEYFILE}"
22    log error "Failed upstream API call ${full_url} with status:
    $http_status_code $api_result" 1>&2
23  log error "Unfortunately, ${SHA_SUM_EXECUTABLE} is required!"
24      log error "The project command requires a sub command"
25        log error "Unhandled project sub-command: $sub_command"
26    log error "Unhandled command-line arguments: $@"
```



Listing B.5: Code Corral / osqueries.rb

```ruby
class OSQuery < Sequel::Model( $DB[:osqueries] )
  many_to_many :projects, class: 'Project', join_table: :project__osqueries
    , left_key: :query_id, right_key: :project_id
  one_to_many :project_osqueries, class: 'ProjectOSQuery', key: :query_id
end
```

Listing B.6: Code Corral / project__osqueries.rb

```ruby
class ProjectOSQuery < Sequel::Model( $DB[:project__osqueries] )
  many_to_one :project, class: 'Project', key: :project_id
  many_to_one :osquery, class: 'OSQuery', key: :query_id
end
```

Listing B.7: Code Corral / project__users.rb

```ruby
class ProjectUser < Sequel::Model( $DB[:project__users] )
  many_to_one :project, class: :Project
  many_to_one :user, class: :User

  def active?
    active
  end
end
```

Listing B.8: Code Corral / project_osquery_user_answers.rb

```ruby
class ProjectOSQueryUserAnswer < Sequel::Model( $DB[:
    project_osquery_user_answers] )
  many_to_one :project_osquery, class: 'ProjectOSQuery', key: :
    project_osquery_id
  many_to_one :user, class: 'User', key: :user_id
end
```

Listing B.9: Code Corral / project_osquery_user_possible_answers.rb

```ruby
class ProjectOSQueryUserPossibleAnswer < Sequel::Model( $DB[:
    project_osquery_user_possible_answers] )
  many_to_one :project_osquery, class: 'ProjectOSQuery', key: :
    project_osquery_id
  many_to_one :user, class: 'User', key: :user_id
end
```

Listing B.10: Code Corral / projects.rb

```ruby
class Project < Sequel::Model( $DB[:projects] )
```

```
  many_to_many :osqueries, class: 'OSQuery', join_table:
    project__osqueries, left_key: :project_id, right_key: :query_id
  one_to_many :project_osqueries, class: 'ProjectOSQuery', key: :project_id

  def self.by_name_dataset( name )
    Project.filter( name: name )
  end
  def self.by_name( name )
    by_name_dataset( name ).first
  end
end
```

Listing B.11: Code Corral / software_versions.rb

```
class SoftwareVersion < Sequel::Model( $DB[:software_versions] )
  def active?
    self.active
  end
  def self.latest_by_name( name )
    SoftwareVersion.where( name: name, active: true ).reverse( :created_at
    ).first
  end
end
```

Listing B.12: Code Corral / user_audit_log_levels.rb

```
class UserAuditLogLevel < Sequel::Model( $DB[:user_audit_log_levels] )
  one_to_many :user_audit_logs, class: :UserAuditLog, key: :level_id
  def self.by_name( name )
    UserAuditLogLevel.filter( name: name.to_s.downcase ).first
  end
end
```

Listing B.13: Code Corral / user_audit_logs.rb

```
class UserPublicKey < Sequel::Model( $DB[:user_public_keys] )
  many_to_one :user, class: :User

  def to_s
    "UserPublicKey[#{id}](#{user.email}:#{name})"
  end
end
```

Listing B.14: Code Corral / user_public_keys.rb

```
class UserPublicKey < Sequel::Model( $DB[:user_public_keys] )
```

```
  many_to_one :user, class: :User

  def to_s
    "UserPublicKey[#{id}](#{user.email}:#{name})"
  end
end
```

Listing B.15: Code Corral / user_whitelisted_regions.rb

```
class UserWhitelistedRegion < Sequel::Model( $DB[:user_whitelisted_regions] )
  many_to_one :user, class: :User
end
```

Listing B.16: Code Corral / users.rb

```
class User < Sequel::Model( $DB[:users] )
  one_to_many :whitelisted_regions, class: :UserWhitelistedRegion
  one_to_many :projects, class: :ProjectUser
  one_to_many :public_keys, class: :UserPublicKey
  one_to_many :audit_logs, class: :UserAuditLog

  def self.by_email_dataset( email_address )
    User.filter( email: email_address )
  end

  def self.by_email( email_address )
    by_email_dataset( email_address ).first
  end

  def active?
    active
  end

  def audit_log( level, message )
    puts "[AUDIT:#{level.to_s.upcase}] #{email} - #{message}"
    add_audit_log( level: UserAuditLogLevel.by_name( level), message: message )
  end

end
```

Listing B.17: Code Corral / 404.erb

```
<html>
  <head>
    <title>Not Found</title>
  </head>
  <body>
    <h3>Not Found</h3>
  </body>
</html>
```

Listing B.18: Code Corral / 500.erb

```
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    <h3>Error</h3>
    <p><%= @error %></p>
  </body>
</html>
```

Listing B.19: Code Corral / control.rb

```
def verify_software_version( hash, return_enumerate )
  software_version = SoftwareVersion.where( hash: hash ).first
  update_required = false
  if software_version
    unless software_version.active?
      update_required = true
    end
  else
    update_required = true
  end
  if update_required
    if return_enumerate == "client" # Client disabled or not found
      halt( 400, "bad client" )
    end
  else
    latest_software_version = SoftwareVersion.latest_by_name(
    software_version.name )
    update_required = true unless latest_software_version ==
    software_version
  end
  if update_required
    halt 200, "update|#{return_enumerate}"
  end
end

if settings.production?
  before do
    halt(400, 'http') if (request.env['HTTP_X_FORWARDED_PROTO'].downcase !=
    'https')
  end
end

before %r|/api/1\.0(?:/.*)?| do
  halt 200, "update|client"
end

before %r|/api/1\.1(?:/.*)?| do
  user_email_address = request.env['HTTP_X_CODE_CORRAL_EMAIL_ADDRESS']
  halt( 400, "no email" ) unless user_email_address

  # Yes we could include the client in the tooling area below, but we want
    to expand
  # tooling into a data driven mechanic eventually rather than hard coded
```

```
    as it
# currently is.
client_version_hash = request.env["HTTP_X_CODE_CORRAL_CLIENT"]
halt( 400, "no client hash" ) unless client_version_hash
verify_software_version client_version_hash, "client"

%w|veracrypt osquery|.each do |tool_name|
  software_hash = request.env["HTTP_X_CODE_CORRAL_TOOL_#{tool_name.upcase}"]
  halt( 400, "no #{tool_name} hash" ) unless software_hash
  verify_software_version software_hash, tool_name
end

auth = request.env['HTTP_X_CODE_CORRAL_AUTH']
halt( 400, "no auth" ) unless auth

code_corral_signature = Base64.decode64( auth )
halt( 400, "empty auth" ) unless code_corral_signature

@user = User.by_email( user_email_address )
halt( 400, "auth denial" ) unless @user and @user.active?

location = Geocoder.search( request.ip ).first
halt( 400, "unable to detect location from #{request.ip}" ) unless location

descriptive_location = "#{location.city} #{location.province} #{location.country}".strip
region_code = "#{location.country_code}_#{location.province_code}_#{location.postal_code}".upcase

@verified_region = nil
@user.whitelisted_regions.each do |whitelisted_region|
  @verified_region ||= whitelisted_region if %r|#{whitelisted_region.region_code}| =~ region_code
end

unless @verified_region
  puts "WARNING: region_code #{region_code} not allowed for #{@user.email}"
  @user.audit_log :warning, "Access DENIED from #{descriptive_location}, which could not matched '#{region_code}'"
  halt( 400, "geo failure" )
end
```

```
76
77    @user.audit_log :info, "Accessing from #{descriptive_location}, which
      matched '#{region_code}' due to '#{@verified_region.region_code}'"
78
79    @verified_public_key = nil
80    @user.public_keys.each do |user_public_key|
81      public_key = OpenSSL::PKey::RSA.new( user_public_key.key )
82      unless public_key.public?
83        STDERR.puts "WARNING user public key is not public: #{user_public_key
      }"
84        next
85      end
86
87      begin
88        @verified_public_key = user_public_key if "#{@user.email}|#{
      client_version_hash}" == public_key.public_decrypt(code_corral_signature
      )
89      rescue OpenSSL::PKey::RSAError
90        next
91      end
92    end
93
94    unless @verified_public_key
95      @user.audit_log :warning, "No public key matched!"
96      halt( 400, "auth failure" )
97    end
98
99    @user.audit_log :info, "Verified public key: #{@verified_public_key}"
100 end
```

Listing B.20: Code Corral / osquery.rb

```
1  post %r|#{API_PREFIX}/osquery/answer/(\d+)| do |project_osquery_id|
2    project_osquery = ProjectOSQuery[project_osquery_id]
3    unless project_osquery
4      @user.audit_log :warning, "Attempted to answer a project osquery that
       does not exist: #{project_osquery_id}"
5      halt( 400, 'no such project osquery' )
6    end
7
8    answer = request.body.read
9    @user.audit_log :info, "ProjectOSQuery(#{project_osquery.id}) Answer: #{
       answer}"
10
11   ProjectOSQueryUserAnswer.create( user: @user, project_osquery:
```

```
    project_osquery, answer: answer )
12
13  possible_answers = ProjectOSQueryUserPossibleAnswer.where( user_id: nil,
      project_osquery: project_osquery ).all
14  possible_answers += ProjectOSQueryUserPossibleAnswer.where( user_id:
      @user.id, project_osquery: project_osquery ).all
15  any_answers_succeeded = false
16  possible_answers.each do |possible_answer|
17    if %r|#{possible_answer.answer}| =~ answer
18      any_answers_succeeded = true
19    else
20      if possible_answer.must_match
21        halt 400, "must match project query #{project_osquery.id}"
22      end
23    end
24  end
25  unless possible_answers.empty? or any_answers_succeeded
26    halt 400, "bad result for project query #{project_osquery.id}"
27  end
28
29  halt 200, 'osquery|ack'
30 end
```

Listing B.21: Code Corral / ping.rb

```
1 get %r|#{API_PREFIX}/ping| do
2   'pong'
3 end
```

Listing B.22: Code Corral / project.rb

```
1 get %r|#{API_PREFIX}/project/list| do
2   "project|list|#{@user.projects_dataset.where( active: true ).all.map{|pu|
      pu.project.name}.join('|')}"
3 end
4
5 before %r|#{API_PREFIX}/project/([^/]+)/([^/]+)| do |project_action,
      project_name|
6   @project = Project.by_name( project_name )
7   unless @project
8     @user.audit_log :warning, "Attempted to access invalid project #{
      @project.name}"
9     halt( 400, 'invalid project' )
10  end
11
```

```
12    @user_project = ProjectUser.where( project: @project, user: @user ).first
13    unless @user_project
14      @user.audit_log :warning, "Attempted to access unauthorized project #{
      @project.name}"
15      halt( 400, "unable to #{project_action} this project" )
16    end
17
18    @user.audit_log :info, "Accessing project #{@project.name}"
19    @user_project.update( last_accessed_at: Sequel.function( :now ) )
20
21
22    unless project_action.to_sym == :release
23      @project.project_osqueries.each do |project_osquery|
24        # TODO deal with stale answers
25        user_answer = ProjectOSQueryUserAnswer.where( user: @user,
      project_osquery: project_osquery ).first
26        unless user_answer
27          @user.audit_log :info, "ProjectOSQuery(#{project_osquery.id})
      Question: #{project_osquery.osquery.query}"
28          halt 200, "osquery|q|#{project_osquery.id}|#{project_osquery.
      osquery.query}"
29        end
30      end
31    end
32
33  end
34
35  post %r|#{API_PREFIX}/project/clone/([^/]+)| do |project_name|
36    unless @user_project.current_encryption_key == nil
37      @user.audit_log :warning, "Attempted to clone an already cloned project
      #{@project.name}"
38      halt( 400, 'project already cloned' )
39    end
40
41    new_encryption_key = $uuid.generate
42    @user_project.update( current_encryption_key: new_encryption_key )
43    @user.audit_log :info, "Cloned project #{@project.name}"
44
45    "project|clone|#{@project.name}|#{@project.repository_uri}|#{@project.
      recommended_disk_space_in_mb}|#{new_encryption_key}"
46  end
47
48  post %r|#{API_PREFIX}/project/open/([^/]+)| do |project_name|
49    if @user_project.current_encryption_key == nil
```

```
50      @user.audit_log :warning, "Attempted to open an uncloned project #{
        @project.name}"
51      halt( 400, 'project not yet cloned' )
52    end
53    old_encryption_key = @user_project.current_encryption_key
54    new_encryption_key = $uuid.generate
55    @user_project.update( current_encryption_key: new_encryption_key )
56    @user.audit_log :info, "Opened project #{@project.name}"
57    "project|open|#{@project.name}|#{old_encryption_key}|#{new_encryption_key
        }"
58  end
59
60  post %r|#{API_PREFIX}/project/release/([^/]+)| do |project_name|
61    if @user_project.current_encryption_key == nil
62      @user.audit_log :warning, "Attempted to release an uncloned project #{
        @project.name}"
63      halt( 400, 'project not yet cloned' )
64    end
65
66    ProjectOSQueryUserAnswer.where( user: @user, project_osquery_id:
        ProjectOSQuery.where( project_id: @project.id).select(:id)).delete
67
68    @user_project.update( current_encryption_key: nil )
69    @user.audit_log :info, "Released project #{@project.name}"
70
71    "project|released|#{@project.name}|"
72  end
```

Appendix C

Example Configuration

Examples provided by finide (2017)

Listing C.1: Get Logged in Users

```
select * from logged_in_users ;
```

Listing C.2: Previous Logged in Users

```
select * from last ;
```

Listing C.3: Firewall Running

```
select * from iptables ;
```

Listing C.4: Firewall Configuration

```
select chain, policy, src_ip, dst_ip from iptables ;
```

Listing C.5: Scheduled Tasks

```
select command, path from crontab ;
```

Listing C.6: Binaries with setuid-enabled

```
select * from suid_bin ;
```

Listing C.7: Active Kernel Modules

```
select name, used_by, status from kernel_modules where status="Live" ;
```

Listing C.8: Services Listening

```
select * from listening_ports ;
```

Listing C.9: File Activity

```
select target_path, action, uid from file_events ;
```