Bearicade: A Novel High-Performance Computing User and Security Management System Augmented with Machine Learning Technology

Taha Al-Jody

High-Performance Computing Research Group
University of Huddersfield
United Kingdom

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

August 2021

I would like to dedicate this thesis to my loving parents, Ali and Nadia who gave me invaluable faith, inspiration and support.

To my sister, family and friends.

In Memory of my brother.

Copyright

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful

Special appreciation goes to my teacher, mentor and supervisor
Dr Violeta Holmes for her dedicated support, patient guidance and
inspiring encouragement.

I would like to thank my family, friends and colleagues
for their for their continued support.

Abstract

Despite the rising development and popularity of HPC systems, there have been insufficient advancements towards the security of HPC systems. The substantial computational power, high bandwidth networks, and massive storage capacity provided in the HPC environment are desirable targets for the attackers.

The majority of educational institution HPC centres provide their users with simple access methods lacking the modern security needs. Thus, accelerating the systems' proneness to modern cyber-attacks. The current implementations of HPC access points, such as web portals, offer users direct access to the HPC systems. Consequently, such web portal implementations affect the HPC system with the same security challenges faced by cloud providers and web applications. Although attempts have been made toward securing HPC systems, most of these implementations are outdated, insufficient with the current security standards, or do not integrate well with modern HPC access solutions.

To address these security issues, Bearicade, a novel High-Performance Computing (HPC) user and security management system, was designed, developed, implemented and evaluated. Bearicade is a data-driven secure unified framework for managing HPC users and systems security. This framework is an add-on layer to an existing HPC systems software, collecting over 50 different types of information from multiple sources within the HPC systems. It offers Artificial Intelligent security solutions with an added usability and accessibility without adversely affecting the performance and functionality of HPC systems. Throughout this study, the security and usability of Bearicade were validated implementing multiple Machine Learning models. It has been deployed over three years as a production system for students and researchers at the University of Huddersfield QueensGate Grid (QGG) with considerable success, protecting the QGG systems from the summer 2020 attacks that has affected many other HPC systems in research and educational establishments.

Table of contents

List of figures

List of tables

Publications

Bearicade: secure access gateway to High Performance Computing systems (Al-Jody et al., 2020)
This conference paper was presented at The 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Some information presented in this conference paper was included in this thesis. Such minor thoughts were paraphrased and structured throughout Chapters 3 and 5. I would like to confirm that I was the main author of this conference paper.

Security Orchestration, Automation and Response (SOAR) in HPC (Al-Jody and Holmes, 2019)
This topic was presented as a poster at the Emerging Technology Conference 2019. I would like to confirm that there was no particular information used in this thesis "adopted from this piece of research".

Chapter 1

Introduction

Supercomputers are also known as High-performance Computing (HPC), a term given to an extremely powerful computer set. HPC is one of the leading technologies in aerospace, automotive, finance, military and sciences fields. It is a necessity for research and focuses mainly on improving computing performance through parallel algorithms, computer architecture and software development. In academia and industry, clusters and grids are the two dominant ways in deploying High-Performance computing.(Pellerin David et al., 2015)

User management systems for HPC is often an afterthought of current administrative practises that fail to provide modern security measures for HPC systems. In the research and education communities, there are no general HPC protection standards. Each organisation has its own policy, which is often validated by certification authority (CA). Because the HPC systems are often behind institutional firewalls, most registered users use Simply Secure Shell (SSH) (Daniel J. Barrett, Richard E. Silverman, 2005) to access given HPC resource. The institutions often hold training events to educate the users on how to access their HPC resources; this is sometimes perceived as a barrier to entry to HPC. However, the client-side software used to access HPC systems are generally insecure or being operated in an insecure environments, such as Putty (Meister, 2007). Usually, several HPC centres (Vacca, 2004) use Private Key Infrastructure (PKI) as their primary authentication method. This approach

has been shown to be reliable. However, even with PKI authentication in place, users with a low level of security knowledge can face security issues.

Most attacks have a life cycle of development. Every step of its development may result in indicators that may be hard for security professionals and system administrators to notice. Hence SIEM systems development was required. SIEM systems provide visualisation of data. Also, they facilitate the collection of data and events in most system components, giving system administrators a massive amount of data that they are able to analyse. Due to the improvements of SIEM systems, security administrators are being swamped with data, which in most cases might be noise, making attackers mission simpler.
The integration of artificial intelligence tools with security systems like SIEM might help with this issue and become the systems administrator's right-hand.
Artificial Neural Network (ANN) could be used to analyse the data from collectors, then perform data classification, identify patterns, and detect anomalies.

1.1 HPC Security

Cyber Security becomes a significant necessity in every scheme to be enforced. Security requirements are a crucial topic for high-performance security systems, and more emphasis should be placed on them.
High-performance computing has received incredible hardware and software development and enhancement. It also faces a rise in the amount of research, industrial or other business uses. In addition, the security aspect and user communication and authentication are not sufficiently developed (Dewayne Adams, 2011).
According to Dewayne Adams, the Chief Technology Officer at Patriot Technologies, high-performance computing safety varies from standard enterprise-level safety.
From the workshop "Cybersecurity for HPC Systems: Challenges and Opportunities" by Sean Peisert in the National Institute of Standards and Technology, the threats of High-Performance Computing Systems could be divided into three sections CIA (Confidentiality, Integrity, and Availability) (Peisert-Lawrence Berkeley National Lab, 2016).

Confidentiality is the vitality and the sensitivity of the data. Integrity focuses on the authenticity of the code and the adequate usage of computational power. Finally, availability concentrates on potentials of attack that might affect the availability of the services, such as Denial of service attacks and overflow attacks.

1.2 Motivation

In the past, security breaches targeting HPC systems tended to be few and far between. Nowadays, HPC systems are facing an increase in vulnerabilities and attacks. Vulnerabilities and weak security implementations may lead to attacks such as phishing, malware, account takeover, data breach, brute-force and many more. In addition, the rewards for attacking such systems are becoming more valuable. Thus, improving the security of HPC systems is a necessity (Kesteren, 2019).

At the University of Huddersfield, there is a range of HPC Systems only protected by custom firewall rules set by the University. There have been various attacks targeting numerous systems at the University of Huddersfield. Specially Queens Gate Grid Systems and High-Performance Computing Research group with attack attempts reaching up to 50,000 attacks per day.
Moreover, a malicious group attacked university data centres in summer 2020 for CPU cryptocurrency mining. The group was able to gain access to various HPC grids by hopping between infected SSH accounts. This cyber-attack has impacted several HPC centres in the United Kingdom, resulting in prolonged downtime in some cases.

Based on the recent meeting of the HPC-SIG Group, which consisted of representatives from the UK's High-Performance Computing Community, there is no universally accepted security framework. The current existing solutions of a security framework for High-Performance

Computing are private unreleased institutional solutions. The university members of the HPC-SIG plan to address the issues raised above.

1.3 Aims and Objectives

The overall purpose of this project is to create a novel, data-driven and secure framework for user and security management enabling autonomous anomaly detection through Machine Learning technologies with the application in High-Performance Computing (HPC) Systems.

In order to satisfy this aim, the following objectives are addressed:

- Investigate current SIEM and SOAR frameworks designed for usage in HPC systems.
- Explore Artificial intelligent algorithms designed for anomaly detection in access and usage of HPC systems.
- Find out AI tools and integrations with SOAR systems.
- Devise new novel secure framework specifically for HPC.
- Deploy the framework in the University of Huddersfield Campus Grid for management of HPC resources and users.
- Allow the University of Huddersfield Campus Grid users to use the framework as the main gateway for the HPC resources.
- Collect different types of data from various sources within the framework related to system security.
- Enhance the security of the novel framework using Artificial Intelligence tools.
- Evaluate the effectiveness of the deployed solution, using the system and users' data, as reliable and secure framework for accessing and utilising the HPC systems.

1.4 Methodology

Methodologies impose a disciplined approach on software development in order to make it more predictable and efficient. The guidelines from the literature will serve as a way to verify that the developed solution leads to the achievement of the research goal. Examples of these guidelines are (i) the definitions of characteristics presented by existing implementations (ii) the functional and non-functional requirement for an HPC portal (iii) the security standards and implementations for web applications and servers.

1.4.1 Software Design Models

To build software products, a planned and systematic procedure should be used. Developing and maintaining the code required to build software is referred to as "software development" (McCormick, 2012). The process of software development may include ideas based on the requirement analysis, an essential design of the software, or a change in requirement through to the final stages of the development of the software. Thus, a flexible and dynamic software design model has to be adopted for this purpose. Some of the software development models commonly used are Waterfall, Spiral, UML and Agile models.

Waterfall

The waterfall model is one of the traditional software design models that define the process for software development into a specific sequence. However many issues have been identified in the waterfall model, commonly acknowledged issues include dealing with change and the fact that flaws are sometimes identified too late in the software development process (Petersen et al., 2009).

Agile

Agile is one of the most recent models to be introduced as a software development methodology, it allows for smaller components of the software to be developed first allowing for the ability to acknowledge changes throughout the process of the development which is happening at regular intervals (McCormick, 2012).

Prototyping

Prototyping methodology requires building a prototype of the software with the essential functionalities included. The prototype is developed further as feedbacks are received. This methodology is appropriate for large projects especially when the requirements are difficult to define or when the project is unique and there are no examples exist (Despa, 2014).

1.4.2 Adopted Software Design Model

To better justify the adopted software design model, a brief overview of the steps carried in this research is outlined as such:

1. An extensive literature review will be carried over the current published work around the access methods and the security around the High-Performance Computing environment. The current existing HPC security methods will be investigated, demonstrating the gaps in security for HPC systems.

2. A unified experimental system will be designed as a test-bed to collect data from HPC systems and users in order to analyse and understand such data and further build the system to be capable of identifying security issues.

3. Experiments will then be conducted to evaluate the experimental system capability in identifying security issues and the capability of allowing further research to be carried in order to advance the security of the unified system contentiously.

4. As part of the experiments, novel security platforms and AI tools will be designed, developed and evaluated using the data gathered over four years.

5. The system will also be evaluated using the University of Huddersfield QueensGate Grid students access to the system and through a set of questionnaires.

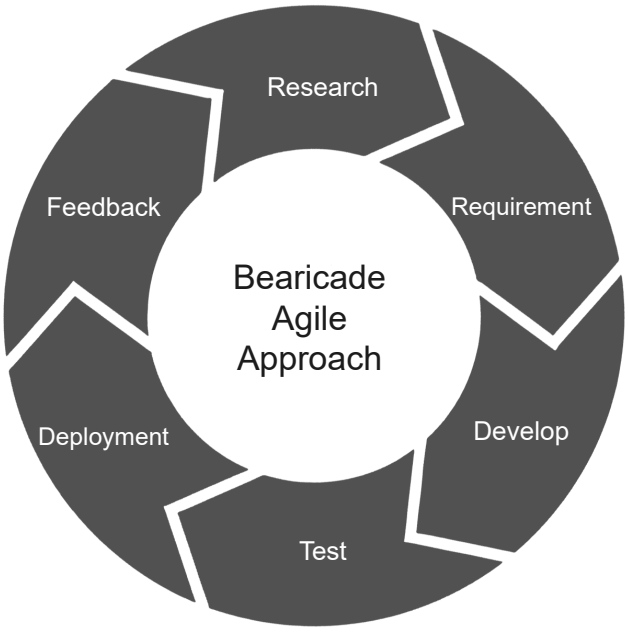

Fig. 1.1 Agile approach for Bearicade's Development

Considering the known stages of the software design models and the overview of the planned steps to be carried in this research, Agile methodology is more suitable than the others for developing Bearicade. As shown in figure 1.1, Bearicade will involve researching and information gathering for what components should be included in Bearicade and their functionality. Following the research design requirements will be devised in order to work on the development of Bearicade. After the development phase, Bearicade will be tested then deployed on the main QueensGate Grid gateway. The feedback will be sought after the deployment through the users' feedback and technical logs on Bearicade.

The Agile methodology is done over several cycles meaning that the phase will be repeated multiple times to ensure such a large project is completed on time with the required

functionality. A more in-depth procedure of the deployment process is explained in chapter Deployment of Bearicade.

1.4.3 Research Methods

Research Methods are a systematic and organised approach to data gathering and analysis in order to extract information (Jankowicz, 2000). Developing Bearicade will be closely tied with Data Science, Mathematics and Information Systems. Therefore, different components of Bearicade could be evaluated through measurements or proofs. Consequently, In order to evaluate the claims and functionality of Bearicade as a framework, a mixed empirical research methods approach has been chosen for this purpose.
Surveys Will be conducted before the deployment of Bearicade and after in order to measure the usability of Bearicade compared to the predecessor access software. In addition, Bearicade users in the classroom will be observed in order to measure the effectiveness of Bearicade and modify the requirements. Finally, data gathered through Bearicade will be visualized and screened then use-cases will involve building Machine learning models in order to evaluate their own accuracy and Bearicade's functionality.

1.5 Summary

In this chapter the motivation, aims and objectives of this research and research methodology were defined. The rest of the thesis is organised as follows: Chapter 2 Literature Review, Chapter 3 Bearicade, Chapter 4 Deployment of Bearicade, Chapter 5 Artificial intelligence-based tools for secure access to HPC resources and Chapter 6 Conclusion and future work.

Chapter 2

Literature Review

In this Chapter, there will be a complete list comparing HPC access solutions describing the requirement and different implementations. The section will explore the security issues and solutions within the different implementations of HPC and distributed systems along with Artificial intelligence solutions.

2.1 Access to HPC Systems

Throughout this section different technologies were investigated for the secure access to and utilisation of the HPC systems. Since HPC systems receive a considerable interest from attackers trying to get access to HPC cluster due to the high computational power that HPC systems offer for its users. Therefore, robust and secure access mechanism has to be implemented in such systems.

Many of the UK's HPC Centres, especially ones in educational institutions, use Secure Shell (SSH) as a standard for accessing HPC systems. The usability of HPC systems is considered to be another limitation to most HPC users.

2.1.1 Secure Shell (SSH)

Secure Shell (SSH) is one of the most widely used protocols for protecting network communications. Since this approach is known for its vigorous history since 1995.

As most HPC centres use SSH as a primary way to interact with the system via a terminal, requiring a basic knowledge of Bash Unix shell and command language (Newham, 2005) to get started with interacting with the system. Bash typically run in a window (terminal) that make use of character encoding methods such as American Standard Code for Information Interchange (ASCII) Unicode Transformation Format (UTF) in order to render text (Kuhn, 2005). Users then read the text and execute commands, and these commands are prone to syntax errors. A graphical user interface solution would improve the ease of use.

SSH consists of two elements server and client, and the whole point of SSH is to encrypt the communication of packets over the server and client. Authentication is one of the client and server's main features, and SSH could be configured to authenticate users via passwords or Public Key Infrastructure (PKI); this will be explored later in this section.

SSH is based on top of Transmission Control Protocol (TCP), meaning that it uses TCP/IP as its transport mechanism. This gives the protocol a varied range of opportunities and challenges.

The server component defaulted to run on TCP port 22, allows clients to access the server remotely rather than being fixed to a local network. On the other hand, this opens up the opportunity for attackers to target servers with several attacks, especially the server will be vulnerable to all attack vectors TCP is facing (Daniel J. Barrett, Richard E. Silverman, 2005).

As a solution for HPC users to access their systems, it is considered relatively lightweight for the system, does not require a demanding setup process, and has little latency and bandwidth issues. However, the users are still required to be knowledgeable in shell command language

depending on the system. Also, the users have full responsibility for not sharing their private key and/or their password with unauthorised individuals.

Secure File Transfer Protocol (SFTP)

Secure File Transfer Protocol (SFTP) is usually used to complement Secure Shell (SSH) functionality by allowing HPC users to access files on the remote HPC storage nodes and easily transfer file between the user's local machine and the main HPC storage nodes (Kruegel, 2004). The functionality of SFTP is similar to Secure File Copy (SCP).

2.1.2 Virtual Network Computing (VNC)

Like SSH, Virtual Network Computing is considered as an HPC access software; it allows users to control the HPC gateway through a graphical user interface (GUI) (Richardson et al., 1998). VNC has a significant effect on the gateway's latency and bandwidth as a result of the GUI capability.

VNC could also be setup to communicate over Transport Layer Security (TLS), hence all communication will be encrypted. To authenticate users, a username and password would be required from the user, and these credentials are typically the same as the system's credentials.

2.1.3 HPC Access Portal

Many web HPC systems portals have been created since the 1990s, when Java Applets became common for full-stack creation, to make it easier for users to access HPC systems remotely. Using a web HPC dashboard is considered the best way of making use of HPC resources (Calegari et al., 2019). Upcoming section HPC Portal Solutions, will show the current implementations of HPC Portal Solutions available. The next section will provide detailed

information about these portals' requirements and analysis of the security vulnerabilities in some of the popular HPC portal solutions.

2.2 HPC Portal Requirements

This section will outline the requirements that qualify a software to be a suitable HPC management system. This section will divide and consider the requirements to two sections, Functional and Non-functional requirements.

2.2.1 Functional Requirements

Functional Requirements are the functions that could be included in the software to operate an HPC system. Even though Functional requirements are mandatory, this section will include some of the non-mandatory functions that have been implemented in many HPC Portals. A 2019 paper on Web portals for High-Performance Computing summarizes two functional requirements to be mandatory for a High-Performance Computing Web Portals (Calegari et al., 2019).

Job Management

This is the core functionality of an HPC system: managing the jobs and the applications executed. This functionality should be able to perform multiple core functions such as submit, monitor, start, stop and pause (Calegari et al., 2019).

Data Management

This functionality complements the Job Management function by enabling HPC centres data to be remotely managed by end-users. "So one of the main challenges for HPC as a Service (HPCaaS) is to make remote data management as flexible as if it were local" (Calegari et al.,

2019). The following functions should be considered in the making of this requirement: “”upload and download files, preview file content, copy, move, rename, delete files, compress and uncompress archive files, browse user data spaces (restricted to users’ privileges), manage file ACLs (Access Control List), monitor quota(s)“” (Calegari et al., 2019).

2.2.2 Non-Functional Requirements

Non-Functional Requirements are the specification of software to judge the operation of such system for HPC users; these requirements are judged by the software’s usability and efficiency. The following section is an example of the requirements that must be up to standards.

Security

Security is a must for all forms of applications, but it’s particularly important when it’s hosting sensitive data, research & development data, or nodes with a lot of processing capacity. More about this feature (see Section Security Solutions for HPC and Distributed Systems).

Usability

Usability is an important indicator of how interactive a system is. Several papers (Calegari et al., 2019; Greenberg et al., 2005; Köhler et al., 2013; Sadowski and Shewmaker, 2010) stressing over the importance of usability, from the usability of developing a Parallel program to the ease of use when submitting a job to a cluster. As for an HPC portal as a whole, the system should have the following elements of usability along with a description of how the element could be measured or/and designed:

- Operability: HPC Portal should be a convenient and straightforward to use. With all of the functionalities are designed in a way that is easy to be found and executed with the least number of steps from the user.

- Learnability: HPC Portal should be used without any further preparation or knowledge. The various types of users can all follow the same basic portal workflow.
- Accessibility: Accessibility should be a fundamental requirement instead of an element of usability; however for HPC portals will be kept as part of the usability. HPC Portal should be available and simple to use for all users meaning that a number of essential design keys should be enforced, e.g. (colour contrast, disability)
- Design: HPC Portals is considered to be the brand image of the provider, e.g. (Cloud Provider, HPCaaS). Therefore, the portal should have an attractive design to reflect the provider's image and impress the users.

Performance

HPC systems are made to be fast in execution, meaning that performance is vital in every scenario. In the backend, every HPC system consists of an HPC job scheduler built on top of the primary Operating system; these are the two mandatory software layers required to have a basic functional HPC system. Since any layer added to these layers might cause an impact on performance, HPC Portals should incorporate solutions that have a minimal effect on the system's efficiency. These solutions could vary, and some of the examples are:

- **Server Separation**: Web portal could set-up their web-server as an example on a separate node database server on another; this would reduce the bottleneck on the main node.
- **Local Cache Storage**: Web portals could store user's temporary data locally in the browser to reduce the frequency of pushing and pulling data from the server, thus reducing the local on the server.

2.3 HPC Portal Solutions

This section will list the most known existing solutions available as an HPC Portal and analyse their security vulnerabilities and implementations.

A survey titled Web Portals for High-performance Computing (Calegari et al., 2019) has compared the existing HPC Portals.

(Activeeon, 2014; Adaptive, 2006; Altair, 2019; Altair Korea Enterprise Computing Division, 2018; Apache, 2016; Atos, 2010; Bright Computing, 2019; Calegari et al., 2019; CyVerse, 2020; Fujitsu, 2010, 2015; Hudak et al., 2018, 2016, 2017; Inside HPC, 2015; Marru et al., 2011; McCormack et al., 1999; Nimbix, 2012; NIST, 1998; Pierce et al., 2014; Platform Computing, 2009; Quintero and International Business Machines Corporation. International Technical Support Organization., 2012; Rescale, 2012; Sampedro et al., 2017; Sims et al., 2001; SysFera, 2014; The Agave Platform, 2013; UNICORE, 1997; Univa, 2019)

#	Portal	Release year	Type
1	UNICORE Portal	1997	Open source
2	WebSubmit Portal by NIST	1998	(D)
3	EnginFrame by NICE/Amazon	1999	Commercial
4	Apache Airavata Django Portal	2003	Open source
5	eCompute by Altair	2003	Commercial (D)
6	eBatch by Serviware/Bull	2004	Commercial (D)
7	MOAB Viewpoint by Adaptive Computing	2006	
8	HPCDrive by Oxalya/OVH	2007	Commercial (D)
9	HPC Pack Web Components by Microsoft	2008	Commercial
10	Compute Manager by Altair	2009	Commercial (D)
11	Platform Application Center (PAC) by IBM	2009	
12	SynfiniWay by Fujitsu	2010	(D)
13	Sysfera-DS by Sysfera	2011	(D)
14	XCS1/XCS2 by Bull/Atos	2011	(D)
15	JARVICE Portal by Nimbix	2012	
16	ScaleX Pro by Rescale	2012	
17	Agave ToGo	2013	Open source
18	ProActive Parallel Suite by ActiveEon	2014	Open source
19	HPC Gateway Appli. Desktop by Fujitsu	2015	
20	Sandstone HPC	2016	Open source
21	Open OnDemand	2017	Open source
22	XCS3 by Atos	2017	
23	Orchestrate by RStor	2018	
24	PBS Access by Altair	2018	
25	Grid Engine by Univa	2019	Commercial
25	Bright Cluster Manager by Bright Computing	2019	Commercial

2.3.1 Open OnDemand

Open OnDemand is an open-sourced web-based client portal for HPC systems based on the Ohio Supercomputer Center OnDemand platform. The project addresses the gap between traditional HPC access methods such as (SSH and VNC) and accessibility of HPC system through the web (Hudak et al., 2018).

The project's source code is available through Github (Center, 2020); the project presents well-documented installation steps. To review this project, the latest version (1.7.14 at the time of writing this) of the portal was installed and set up on a cloud server instance running Centos 7.6. The portal had a number of functionalities, such as a web-based SSH client, file-manager, and job manager. The portal uses Basic access authentication by default with support for Shibboleth and CAS Single Sign-On and support for second-factor authentication and OpenID through third-party integration. There was no implementation of data-gathering methods in the project. However, support for Google Analytics was provided.

By analysing the portal and its source code, there were a number of security issue presented; some of these issues will be listed below.

Third-party Libraries

The project makes use of a number of libraries in order to achieve it's functionality, some of these libraries are outdated and present the system with a number of security issues. As an example, (A) jQuery (v3.4.1) (B) Cloud Commander (v5.3.1): provides the functionality of SSH and file-manager through the web. Some of the vulnerabilities proposed by libraries A & B will be listed below:

- **Authentication Bypass (Library B)**
- **Cross-site Scripting (XSS) (Library B & A)**
- **(Regular Expression) Denial of Service ((Re)DoS) (Library B)**
- **Remote Memory Exposure (Library B)**

Inadequate Implementation

Along the vulnerabilities listed above, there is more security risk that is present within the project itself:

- **Authentication Credentials Capturing** : as the portal makes use of basic access authentication by default; this authentication mechanism encodes the username and password pair to base64 format, the web application then passes these credentials in the request header, allowing an attacker within the same network of the user to capture their credentials using Man-In-The-Middle attack as an example.
- **Header Misconfiguration**: cross-domain configuration such as Access-Control-Allow-Origin are misconfigured, allowing attackers to access requests through arbitrary third-party domains. In addition, X-Frame-Options and X-Content-Type-Options are not included, allowing possible ClickJacking and XSS attacks.
- **Misconfigured Cookie**: cookies initiated by the portal are lacking SameSite attribute, which is a protection against cross-site request forgery, cross-site script inclusion, and timing attacks.
- **Inessential HTTPS**: HTTPS is not required in order for the portal to function, presenting the users with a high risk of website tampering and eavesdropping attacks.

Demonstrating XSS attack on Open OnDemand

From the vulnerability disscued above from library B, a user is able to create a file/directory with an XSS payload such as *"><svg onload=alert('XSSpayload');>* as the name, this is then reflected to all users accessing the same directory containing that payload as demonstrated in figure 2.1.

2.3.2 Bright Cluster Manager

Bright Cluster Manager (BCM) by Bright Computing offers a complete solution for HPC centres to build and manage clusters and allow users to submit and monitor jobs. Like most

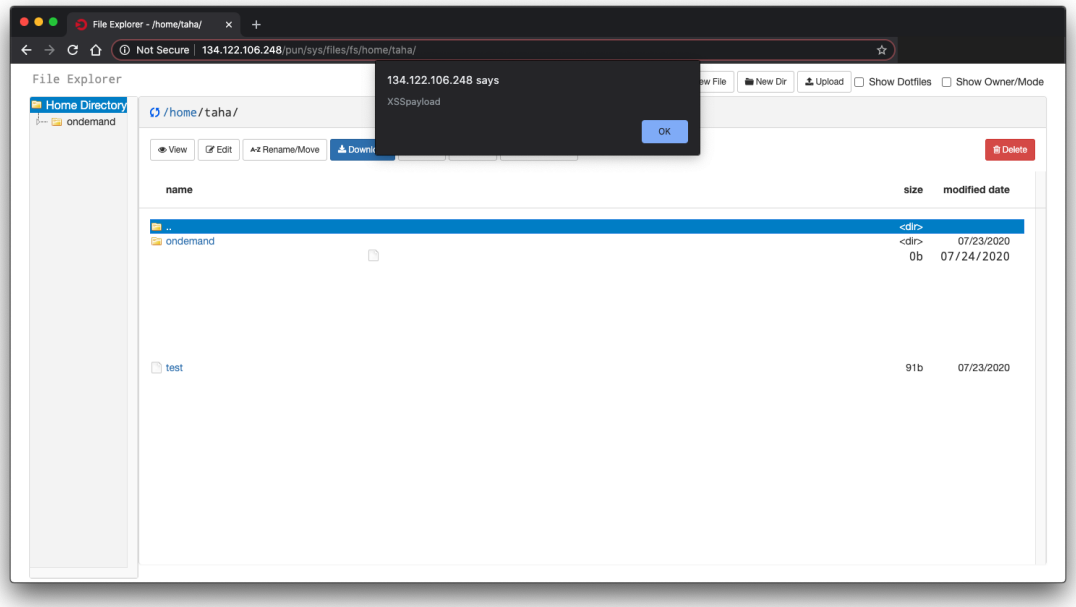

Fig. 2.1 Reflected XSS demonstration on Open OnDemand portal

HPC portal solutions presented in table 2.1, BCM is a commercial product meaning that the software is proprietary. Therefore, information gathered about the software were mainly from available manuals, documentation, and a live demonstration of a software (Bright Computing, 2019).

BCM dashboard has a monitoring functionality that features some measurable information, mainly hardware gathered information such as usage and jobs data. BCM does not seem to feature security-related information.

BCM's developer manual describes a read-only Representational State Transfer (REST) Application Programming Interface (API). However, this implementation uses basic access authentication, which comes with critical security issues discussed in 2.3.1.

2.4 Security in HPC and Distributed Systems

High-Performance computing is a dominant tool for research in academia and industry (Pellerin David et al., 2015), the increased development in HPC has resulted in providers

being a desirable target to intruders (Peisert, 2017). In addition, the following are the main aspect luring attackers to HPC: (1) substantial computational power (2) high bandwidth network (3) massive storage capacity.

In the beginning, when HPC started being adopted for usage in research and education, security which is one of the most significant concerns for both users and HPC system maintainers, was not a significant problem. Nowadays, due to the extensive developments made towards HPC portals. HPC vendors, such as Institutions or HPCaaS, are moving toward delivering online interaction with their systems, such as HPC Portals, to their consumers and clients. Users are provided with an easy access to a pool of high-performance computing services through HPC systems. This access requires permission to use the internal network, which is protected by security measures that ensure data protection. (Prout et al., 2016). Also, HPC portals are made available remotely and accessed through the web. Thus, currently facing the same security challenges being faced by cloud providers and web application.

There have been concerns about the security and privacy of the data in the cloud especially from the business and governmental sectors due to the sensitivity of the data especially if the data is considered to be a national assets (Armbrust et al., 2010; Chang et al., 2011). Studies showed that there is a direct link between data protection laws and the interest in cloud computing (Eldred et al., 2015, 2016). Another research proves that is a huge ambivalence with the adoption of cloud computing due to the security, legal and privacy challenges that cloud computing raises (Khajeh-Hosseini et al., 2010).

2.4.1 Security Challenges

Report made by Ponemon on security automation (Ponemon, 2013), shows that 88% of data is stolen in minutes via cyber-attack or data breaches, 85% intrusions have not been disproved until weeks later; also, the average mitigation time was 123 hours.

Another report written by Verizon (Verizon, 2019) supports Ponemon report, showing 51% of compromises happens in seconds, 36% in minutes. In addition, 88% of Data exfiltration happens in minutes. Moreover, 85% of threats are not discovered until weeks later.

Nowadays, all attackers need is an exploitable vulnerability in any component in any layer of a system, and they can access the system. On the other hand, attacks are receiving more developments, increasing the number of zero-day exploits and malware that are able to dodge most firewalls and software alike and able to spread more, making them persist longer.

Security Challenges in HPC

The Security of HPC systems has been a concern and a path of developments since the 1998 (Campbell and Mellander, 2011; Foster et al., 1998; James D. Ballew et al., 2015; Malin and Van Heule, 2013)

Most common system vulnerabilities are still effective in HPC systems. However, due to an HPC system's characteristics, mitigations for those vulnerabilities are not effective and lack efficiency (Apostal et al., 2012; Simakov et al., 2018).

There has been average progress done into researching security challenges in HPC portals. There is however, a paper (Eldred et al., 2016) that reports a case study exploring security-related queries around protecting sensitive data and the security decisions, as a result of a survey conducted with a number of HPC organisations members. The papers show doubts about data confidentiality in general, including where the data are stored, the authentication and authorisation, and other security-related questions.

A paper from 2013 (Malin and Van Heule, 2013) has proposed Continuous Monitoring for HPC systems. Continuous monitoring is a long-established method for desktop and mobile operating systems that ensures constant and uninterrupted cyber protection via multiple ways. The papers by Los Alamos National Laboratory HPC (Malin and Van Heule, 2013) have claimed that the organisation has started developing continuous monitoring for HPC. Considering that an attacker might attempt to alter multiple vectors such as firewall, system files and services, the paper has given a hypothetical approach to achieving a continuous monitoring system. The approach consisted of the following elements:

- **Firewall Monitoring**: a host-based firewall with a continuous test running to ensure the firewall rules have not been altered or turned off.
- **Logs Monitoring**: constant monitor of logs to detect unauthorised changes in the system. Orchestration tools could be used to revert the system to an optimal state.
- **Services Monitoring**: services are constantly monitored for changes.
- **Security Tools Monitoring**: constant monitoring for security tools to check if the tools are functioning properly.

A study (Prout et al., 2016) suggested using Netfilter, a system-level feature, to create a user-based firewall to administer the connections within the internal network of an HPC system: this will allow administrators to implement a set of rules. This study has focused on the impact of firewall rules on the performance of the HPC system.

According to multiple papers (Gantikow et al., 2016; Higgins et al., 2016; Priedhorsky and Randles, 2017) authors proposed lightweight and secure solutions for HPC by containerising middleware, environments, or software stacks. Such implementations do not consider exploits resulted from the more advanced vulnerabilities such as Meltdown and Spectre (Kocher et al., 2018; Lipp et al., 2018). A paper discussed the proposal of a security balance method for HPC. The paper (Chen et al., 2017) scrutinised the traditional barrel/bucket theory and proposed a new barrel theory. The theory consists of multiple barrels on the inside of the other with each of the barrels representing the following as shown in figure 2.2: "data security, system security, access control, application security, network security and physical security".

The study suggests security methods for each of the layers (buckets) and security measures for the layers based on the new theory. The results are summarized in table 2.2.

A recent study (Luo et al., 2019) has emphasised the importance of log auditing in strengthening security for HPC systems. The study has listed multiple types of attacks in HPC systems, and the paper reviews defence strategies for HPC systems.

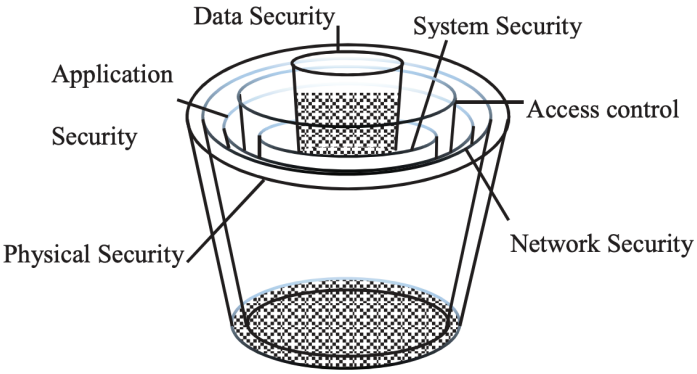

Fig. 2.2 New security balance method for HPC using the barrel theory (Chen et al., 2017)

Table 2.2 Security Layers (Chen et al., 2017)

Layers	Methods
Data	Content Auditing, Transfer Protocol
System	Back-up, Policies, Firewall
Access	Authentication, Authorization
Application	Anti-Virus
Network	Firewall, Network Standards
Physical	Isolation, Monitoring

However, there are no current publications covering the effectiveness of measuring IP reputation, analysing users' behaviour and other intelligent techniques within the HPC environment.

2.4.2 HPC Systems Performance and Security

High-Performance Computing systems, by definitions, are systems optimised for high-performance. A factual statement is that the security of HPC is not being considered differently than standard computing except for when security is antithetical with performance (Peisert, 2017).

Red Hat has published an article (Red Hat, 2017) describing the performance impacts of Meltdown and Spectre (Kocher et al., 2018; Lipp et al., 2018) which are hardware vulnerabilities that affected Intel processors, IBM POWER processors, and some ARM-based microprocessors. The initial impact on performance was 1-20%. However, after optimising the patches, it has been reduced to 1-8%.

A 2018 paper (Simakov et al., 2018) examined the performance impact on HPC systems when applying those patches for Meltdown and Spectre (Kocher et al., 2018; Lipp et al., 2018). The paper has found that some HPC application decreased in performance up to 74%.

Another paper about user-based firewall (Prout et al., 2016) demonstrated no performance issue while using a single-threaded Netcat (Kanclirz and Baskin, 2008) to transfer data. However, once the test was repeated with ten threads, with 1000 connections each, the test showed that the speed has halved. Two other papers have tested the performance of Netfilter/iptables (by Harald Welte, 2000; Purdy, 2004), the results showed minimal bottleneck, especially when handling a large set of rule and on gigabit networks (Hoffman et al., 2003; Kadlecsik et al., 2004).

2.5 Security Solutions for HPC and Distributed Systems

Security tools are becoming more prevalent at all layers of the infrastructure, including programme, network, host, and even client. Firewalls, rules-based network scanners, anti-virus, IDP, and IPS are the most commonly used methods, as will be discussed later.
With these varieties of security tools protecting most components of a system, security professionals "must be at an advantage". However, that is not the case, and infiltrators are succeeding more often with more attacks.
From table 2.3 taken from a paper (Chugh and Chugh, 2010), security of HPC systems could be divided into 3 layers (i) User/Application (ii) Communication (iii) Resource, each of these layers are defined by it's discipline. The authors suggest that existing technologies could be applied to each discipline successfully. However, each of these technologies will have constraints on HPC systems.

This section will review existing solutions to monitor and prevent cyber incidents in other Infrastructure.

Table 2.3 Security discipline at different layers of security on HPC (Chugh and Chugh, 2010)

Security Layer	**Discipline**
User/Application	Authentication Authorization Confidentiality
Communication	Privacy Message Integrity
Resource	Secure Logging Assurance Manageability

2.5.1 Elemental Technologies

Elemental Security technologies are the essential practices for securing HPC systems. The following practices are used in many HPC providers contributing as a base layer of security (Bulusu et al., 2018).

Operating System Security Hardeneing

This practice is the process of hardening the core of the HPC systems, ensuring that directories, users, and services are set up with the correct level of privileges and permissions. Examples of these hardening processes are:

- **Password Complexity Requirement**: a policy should be implemented to enforce the requirements of a password; policies may include the length of the password, complexity i.e. (special characters, numbers, alphabets). Password requirements could increase the entropy of passwords (Marquardson, 2012)
- **Password Expiration**: setting a regular password change policy might not be a way of preventing attackers from compromising passwords. However, it could definitely help in reducing the potential damage from a compromised password (Marquardson, 2012; Zhang et al., 2010).

- **File Permission**: Unix systems implements the owner, group, and other to files and directories, meaning that every file has a single owner (Fenzi and Wreski, 2000). Correct file permissions ensure that users have no access to crucial system files.
- **Services and Packages**: Additional unwanted services and packages open up opportunities for potential attacks, as an example, Red Hat Enterprise Linux 6 contains over 1000 packages and services, most of which are not a necessity for HPC systems (Dheshmukh and Mahalle, 2014). Over time packages becomes vulnerable, leaving unpatched vulnerabilities in the system.
- **File Integrity**: ensuring the integrity of files are a crucial step in preventing many attacks that typical anti-virus software would not be able to distinguish from a legitimate use. An example of this is the recent Dirty Copy-On-Write (COW) vulnerability (Alam et al., 2017; Red Hat, 2016).
- **Log Management**: collecting logs from different services and system components is an imperative step for auditing security issues on a system (Vaarandi and Pihelgas, 2014). Logs will contain the activity of services, users and security events such as elevated commands. A 2019 vulnerability has exploited Sudo version $<$1.8.28, allowing a user to run a command as an arbitrary user (MITRE, 2019).
- **Firewall**: firewall rules offers enhanced network security. Rules should filter inbound and outbound activity, limiting the system's exposure to external and internal network attacks (Prout et al., 2016).

Authentication

Authentication is of vital importance in protecting HPC systems from intruders. As explained previously in section (Access to HPC Systems), there are multiple ways of accessing HPC systems. Similarly, there are many methods for authenticating users; below are most of the popular authentication methods:

- **Password Authentication**: using a password only to connect to the systems.

- **Public Key Authentication**: client provides a key pair to the system where it gets checked with a list of allowed keys.
- **Pluggable Authentication Modules (PAM)**: different modules could be used as a way of authenticating users, e.g. (pam_ldap and pam_mysql).

Regular Security Auditing

One of the most significant security issues is the lack of regular security audits (Rasheed, 2014). However, auditing requires personnel's familiarity with analysis tools and knowledge of the current vulnerabilities. For HPC systems, patches could hugely impact performance, as explained in section: (HPC Systems Performance and Security).

2.5.2 Security Information and Event Management System

Security Information and Event Management Systems (SIEM) provide Security Information Management (SIM) such as log management and activity reports. Also, they provide Security Event Management (SEM) including real-time devices, network, and user's activity monitoring.
SIEM provides four main functionalities but is not limited to (Swift, 2006):

- **Log Management**: A smart logging server that logs from all nodes attached to a system, the log should be reported to the SIEM System to be visualised for fast response. Logs are collected from all different devices and software, such as authentication server and user activity.
- **Intrusion Detection**: Artificial Intelligence automates the process of going through logs and activity in order to detect suspicious behaviour within the system
- **Incident Response Automation**: Incidents should be reported to system administrators through some way of communication.

- **Forensic Investigation Management**.

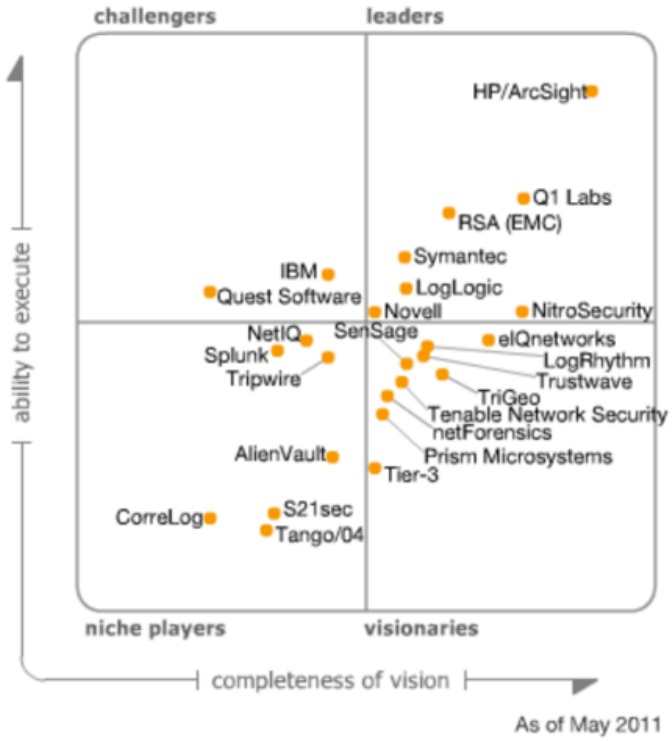

Fig. 2.3 Magic Quadrant for SIEM Evaluation Criteria Definitions

Figure 2.3 (Nicolett and Kavanagh, 2011) above represent the variety of Security Information and Management Systems. There are 24 systems represented in a graph to show SIEM systems' ability compared to their marketing strategies.

Regarding the ability to execute considered product in terms of completeness, features, capabilities, and other services provided, the highest was Hewlett Packard's SIEM system ArcSight which offers a wide range of features and cybersecurity tools. However, the system has no support for High-Performance Computing System and does not include Artificial Intelligence tools.

2.5.3 Security Orchestration Automation and Response

In 2017, Gartner, a global research and advisory firm, invented the term SOAR (Stan Engelbrecht, 2018), to describe a system with layers of security tools stacked together. These

tools allow the system to collect data from information about the security of a system from different sources, detect threats, prioritising attacks, then to respond to these threats without human input.
SOAR systems are very close in functionality to SIEM systems with the added functionality of orchestration and automation.
Through the integration of Artificial Intelligence to existing SIEM systems to apply human intelligence and machine learning would strengthen the Intrusion prevention in the system. This solution would be beneficial for systems with a large amount of log data generated hourly, such as in HPC systems.

In (Donevski and Zia, 2018), the author emphasises the importance and the demand on security automation and management of anomalies, stating that the SOAR system is part of the equation.

A substantial proof that SOAR systems are at the peak of demand and maturing, Splunk acquiring Phantom, FireEye acquired Invotas, IBM acquired Resilient, and recently Microsoft and Rapid7 acquired two more security companies. Most of these companies specialise in SOAR systems (Jon Oltsik, 2018).

2.6 Artificial Intelligence Implementations in Security

The term "artificial intelligence" (AI) existed for more than 60 years, back when John McCarthy, a mathematician and also known to be the creator of Artificial Intelligence, defined a machine that could do what humans are able to do (Shannon, 1956).

With all the advancements that have been made towards Information Technology (IT) presenting internet users with convenience and accessibility in accessing diverse information, these advancements have brought a variety of cybercrimes (Clough, 2015). Cyberattacks present a global threat to computer systems rapidly, especially nowadays, where almost everyone could have the knowledge and tools for attacking computer systems. Traditional security software and Conventional fixed algorithms, such as detectors, are becoming insufficient in monitoring

and protection systems. There is a need for a more astute security implementation capable of differentiating normal and abnormal activities, also, able to react and make decisions in a timely manner (Dilek et al., 2015). This scarcity could be satisfied with innovative solutions such as Artificial Intelligence, giving the learning capabilities in monitoring, detection, and making a decision that is similar to a human's decisions (Nogueira, 2006; Tyugu, 2011). With all of the possibilities offered within Artificial Intelligent solutions such as Neural Networks, Machine Learning, Pattern Recognition, etc., there is a huge potential in such solutions in improving the security of computer systems (Dasgupta, 2006; Patel et al., 2012; Xiao-bin Wang et al., 2008).

As previously mentioned, AI implementation could perform human cognitive abilities and could have sensory functions such as learning, analysing, and reasoning, thus detect then categorise normal activities and differentiate them from abnormal behaviours.

Survey papers (Berman et al., 2019; Buczak and Guven, 2016; García-Teodoro et al., 2009; Nguyen and Armitage, 2008; Sperotto et al., 2010; Wu and Banzhaf, 2010) have mentioned and described the usage of Machine learning in cybersecurity and explained how it solves big issues of cyber problems.

Some papers have explained the usage of machine learning (ML) or its deep subset learning (DL) in cybersecurity for particular applications such as intrusion detection, malware, and spam detection (Apruzzese, G.; Colajanni, M.; Ferretti, L.; Guido, A.; Marchetti, 2018; Xin et al., 2018). In (Wickramasinghe et al., 2018) the authors explain the usage of DL in defending cyber-physical systems. For the Internet of Things (IoT), there have been achievements done in systems security ML (Al-Garadi et al., 2018).

2.6.1 Artificial Intelligent Intrusion Detection Systems

National Institute of Standards and Technology (NIST) describes intrusion as the attempt of compromising confidentiality, integrity, or availability (CIA). An intrusion Detection

System (IDS) is a hardware or software that can monitor computer systems or network communication to detect signs of such an attempt (Bace and Mell, 2001; Liao et al., 2013).

There have been several research publications proposing implementations of different AI solutions to build an enhanced IDS systems. In (Linda et al., 2009), the authors presented an IDS Neural Network model able to detect intrusion attempts with a 100% detection rate. A similar paper has shown an increase in the detection rate in mobile IDS by reducing false positives using Neural Network (Barika et al., 2009). Another paper has shown an implementation of an anomaly IDS using Neural Network by training the model with the KDD 1999 Data; the model was successful in learning packet behaviour (Al-Janabi and Saeed, 2011). Using the same dataset, a paper (Ferreira et al., 2011) has presented another Neural network-based IDS with an accuracy of 99.98% detection rate. Another paper with a Neural network-based IDS using the KDD 1999 dataset has seen a 100% detection rate similar to conventional IDS. However, it has produced a performance increase in detecting Denial of Service attacks (Barman and Khataniar, 2012). There are more papers that have presented similar implementations (EshghiShargh, 2009; Ferreira et al., 2011; Hassan, 2013; Herrero et al., 2007; Jongsuebsuk et al., 2013; Ou et al., 2011; Shao-jing et al., 2011; Shosha et al., 2011; Zhang et al., 2011).

2.6.2 Artificial Intelligence in Access Control

Authentication is a critical step in authorising users to access protected information and resources safely. Throughout the years, there have been many methods used to authenticate users. Text passwords are the most common authentication method. Since 1979 a research of over forty years old showed that passwords come with large security risk (Morris and Thompson, 1979). However, since then, there have been different methods of access control. Table 2.4 shows the ontology of different authentication methods that have been used or could be used to assist in authorising users.

There is a continuous demand in finding new means of authenticating user's identity (Purgason and Hibler, 2012). This subsection will cover related work where some of the authentication

Table 2.4 User Authentication Methods Ontology

Possession Based	**Knowledge Based**	**Behavioural Biometrics Based**	**Physical Characteristics Based**
Keys	Password	Signature	Fingerprint
Universal 2nd Factor	Secret Questions	Voice	Face
Identification Document		Input Device Behaviour (mouse movements, keyboard strokes)	Iris
Multi-factor Authentication			
Smartcards			
Device Properties			

methods shown in table 2.4 are assisted with Artificial Intelligence to improve its performance and increase the general security of access control.

There has been a number of research papers published on how to collect and analyse users behavioural biometrics data to be used as an authentication method or as a second-factor authentication. Although some papers have shown successful implementation of identifying imposter and differentiating between individuals without the use of AI by analysing rhythms in users typing (Bergadano et al., 2003), recent studies have shown that the newer generation of malware and bots are capable of performing synthetic attacks (Serwadda et al., 2011). Therefore, there is a need to implement an intelligent system to counter these types of attack.

Artificial Neural Networks (ANN) were used in a study to build an authentication and discrimination system by recording keystrokes and the time interval between presses of users when inputting their passwords; this was then converted into an RGB histogram and train the ANN with this data. The system was successful in identifying 90% of legitimate attempts and 90% of frauds (Alpar, 2014). Another paper had a similar implementation for a web application where the users' password input properties such as timing interval and the duration of keystrokes were used to train an ANN. This implementation has produced an average of 1% error rate. Similarly, in (Purgason and Hibler, 2012), the authors have produced encouraging results in collecting and analysing behavioural data in order to authenticate user's identity.

In a 2019 study, authors have discussed the importance of artificial intelligence in user authentication. Authors have developed a program to gather data from a small number of users accessing the website and recording (i) mouse click timestamp (ii) coordinates of the mouse and browser window (iii) URL frequency related information. The data was then fed to multiple machine learning algorithms and showed an increase in the accuracy of classification compared to other studies (Kogos et al., 2019).

With the increased usage of internet in users' daily life (We Are Social, 2020), there is enough data to be collected from their users' interaction with a web page in order to identify each user uniquely (Abramson, 2014a,b; Abramson and Aha, 2013) or partially (Fridman et al., 2013, 2017). This has not been implemented in HPC systems, with the potential of preventing unauthorised access. Implementing a system to collect the required data from the users and use it to identify threats would be a contribution to the community.

2.7 Shortcoming of existing HPC Security Practices

High-Performance Computing Systems are used to for computationally and data intensive applications, meaning that any performance degradation caused by security implementations such as IDS/IPS are undesirable. In addition, HPC systems are typically run in critical industries, process valuable information, and serve certain important customers. Due to this, High-Performance Computing environments represent a unique paradigm when compared to other computer systems environments (Bridges et al., 2002). Backing up this statement, HPC systems have distinct modes of operation, in addition to that the exotic applications used by the users which tend to be open to users raising a high security threat (Peisert, 2018).

According to HPCSec, a security specialist for HPC environments, HPC is a niche area where most security experts have not had the experience to work with HPC technologies. Also, security practices are not applied within the HPC field, and traditional security tools are not built to be suitable for HPC systems (HPCsec, 2019).

The University of Huddersfield's High-Performance Computing QueensGate Grid (QGG) receives a large number of SSH login attempts on a daily basis. For example, table 2.5 shows the number of failed SSH login attempts recorded in a 24-hour timeframe per country; the total number of attacks was 46890 from 9378 different IP addresses, all of which were banned from further attempts. During the 24-hour timeframe, login attempts were limited to 5 attempts per IP address every 15 minutes, meaning this number could be vastly increased if such limitation did not exist.

Table 2.5 Failed SSH login attempts on QGG during a 24-hours timeframe

Country	**SSH Attempts**	**# IP Addresses**
China	18145	3629
United States	4850	970
France	3235	647
Brazil	1830	366
India	1645	329
Germany	1405	281
Singapore	1400	280
South Korea	1315	263
Other	13065	2613
Total	**46890**	**9378**

Looking at the literature, there has been little achievements in ML for security systems in distributed systems, and no advancements in security of HPC.

2.8 Shortcomings of the Existing Security Implementation for HPC

Throughout reviewing the literature, it was proven that attacks have a life cycle. Attacks such as breaches take years to discover, especially in large organizations (Verizon, 2020). Based on the literature review conducted of the current state of HPC systems in terms of security and access methods, the following shortcomings are apparent:

1. SSH is used by most HPC centres and has not advanced in security; it relies on a trusted-host authentication mechanism (PKI) by blindly trusting the host providing

the keys, making it an insecure practice hence the May 2020 attack where a malicious group targeted academic HPC centres in order to mine cryptocurrencies. The group was able to get access to several HPC grids by hopping between compromised SSH accounts.

2. Portals lack the automation in managing HPC security. Some portals also lack basic modern security requirements, as proven in the Open OnDemand portal.

3. No Artificial intelligent based implementations to autonomously manage security within the HPC environment.

4. No standard framework that was developed for HPC systems to address the issues stated above.

The gaps in knowledge identified above demonstrate a lack of a unified approach across the UK research and Higher Education institution for secure access to the HPC systems. There are developments of HPC portals that attempt to migrate security issues, Nevertheless, they lack in automating the task of detecting anomalies of users' activities using AI tools.

In chapter 3, the Bearicade system is proposed to overcome the shortcomings of the existing security and access systems.

2.9 Research Questions

There a three main research questions that need to be addressed in this research:

- How can the technologies and data provided by this research contribute to the development and implementation of SOAR system in HPC?
- To what extent such an implementation can protect HPC systems and authorised access?
- Will such implementation help in the Management and the usability of HPC systems?

In order to address the research questions and validate and evaluate the proposed novel SOAR framework for HPC system security, a combination of quantitative, qualitative and experimental empirical research methods will be used. Surveys of users' experience will be used to gather and analyse qualitative data on the system's usability. Data gathered from the experiments of SOAR implementation and deployment of AI tools in the existing HPC systems will be used to validate and evaluate the proposed novel system for enhancing security in HPC systems.

Chapter 3

Bearicade

3.1 Introduction

Security Orchestration Automation and Response (SOAR) has a potential to counter emerging security concerns in HPC systems. The SOAR principles were used when developing a novel framework - Bearicade to address the concerns in HPC systems security.

"When designing an HPC portal, as for any other software, use cases have to be thoroughly studied and specified. All HPC portal functions and features are not always used or even used at all, depending on the use case. Not all HPC portal users have the same objectives or use the same workflows. The same applies to organizations delivering the HPC service" (Calegari et al., 2019)

Bearicade is built on well-known browser features, and it allows for smooth management and connectivity to HPC systems without requiring specialised user training. The Bearicade system's architecture is depicted in Figure 3.1. The two-factor authentication used for user registration and authorization is either a time-based one-time password or an universal second factor (U2F). System protection is accomplished by continuously tracking, analysing, and interpreting data, such as user interaction, server requests, devices, and geographical locations, among other things. System managers will be able to see unusual user behaviour. and will facilitate prompt actions in order to mediate the threats. Also, the system does

not use SSH as its main external access gateway to the system. Instead, it uses the web dashboard as a gateway.

3.2 Software Architecture

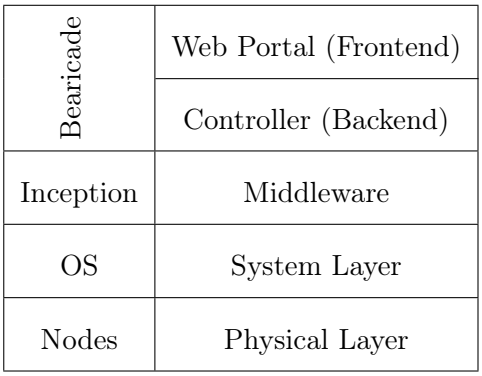

Fig. 3.1 System Architecture Layers for Bearicade (Al-Jody et al., 2020)

In order to demonstrate the performance and the functionality of the proposed framework and to gather information required for the system to be implemented and developed, a proving ground has to be set up in the form of an HPC system test-bed. Bearicade will be deployed and used in the University of Huddersfield's QueensGate Grid (QGG) main HPC management system. Bearicade will be implemented on top of the three HPC systems layers, as seen in figure 3.1, similar to other software access methods for HPC systems; the three base layers are the physical layer, the system layer, and the middleware layer.

3.2.1 Physical Layer

The physical layer is the base hardware layer of any HPC systems, representing the backbone of HPC systems granting the powerfulness for the system. In a typical HPC environment, grid utilizes the power of large sets of computing resources to meet the needs of compiling intensive high-performance computing, as represented in figure 3.2. Grids consists of multiple clusters that could be in numerously geographical locations, each of the clusters is a set of nodes often connected. The clusters mainly consist of a single head node and a number

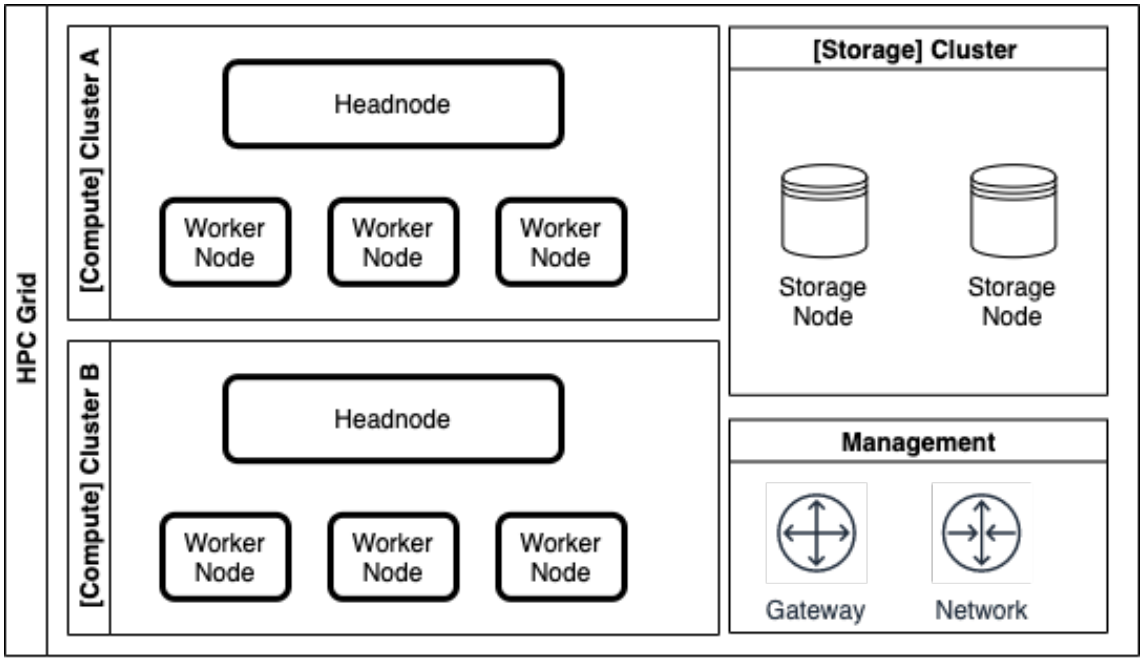

Fig. 3.2 Typical HPC Grid Components

of worker nodes. Clusters and nodes could have a variety of functions contributing to the operability of an HPC grid, i.e. compute, network, management, backup, and storage. Customarily grids should be connected via a network, enabling the ease of managing all nodes within each cluster.

Bearicade is designed to run in a separate node(s), meaning that it does not affect the current configuration of any HPC systems. Bearicade could be deployed on a bare-metal machine or container ready machine as will be explained in the Deployement of Bearicade section.

3.2.2 System Layer

System layer is the base Operating System (OS) layer of an HPC system, a number of HPC centres uses RedHat Linux, including RedHat Enterprise Linux (RHEL) and CentOS, which is compiled version of the open-sourced RedHat Linux. Over the past years, a number of HPC-ready OS made available such as ROCKS (Rocksclusters, 2018) being one of the last surviving OS as of 2018 is also based on CentOS. Another popular Linux distributions are SUSE Enterprise Linux (SLE) and Debain based distributions such as Ubuntu. Both RHEL

and SLE uses RPM Package Manager (RPM) to build sources for HPC systems (OpenHPC Community, 2018).

3.2.3 Middleware

At the University of Huddersfield, deployments of High-Performance computing clusters are automated through the use of an orchestration tool. This tool is called Ansible, which is an open-source software provisioning, configuration management, and application-deployment tool (Red Hat, 2019).

An ansible playbook has been developed at the University of Huddersfield (Higgins et al., 2018), the playbook uses OpenHPC, which provides all HPC components in a package (OpenHPC Community, 2018). For the deployment and provisioning of the nodes in every cluster, the playbook uses Extreme Cluster Administration Toolkit (xCAT), which is an open-source deployment and provisioning middleware (XCAT, 2018). By providing a config file and a machine file to include the list of nodes, sensible playbill copied install, boot, and manage all nodes (Clusterworks, 2018).

3.2.4 Bearicade Stack

Bearicade is a full-stack application that drives the 3 HPC layers (Physical, System, and Middleware), as with full-stack components, there are two layers to the stack. The controller, which is the backend of Bearicade being the closest to the HPC system layer. It handles operations with the system components below it as shown in figure 3.2. The second layer of the stack is the Web portal which is the frontend of Bearicade and the closest to the users; it handles users' interactions and uses the controller to feedback to the system.

Data Storage

Bearicade is a data-driven framework, meaning that without the information collected from the users, it would not be able to achieve its purpose. Due to the large number of users of Bearicade and the number of data generated (i.e. activity, logs), Bearicade requires a reliable, scalable, and most importantly, secure database. Bearicade makes use of two databases under the same server, the first of which is an authentication database that stores users login credentials, sessions, system configuration, and attempts. The second database is used to store all activities of the users; examples of these data include: how the users interact with Bearicade, what commands and activity they perform, and classification data. MySQL is the database server used, is an open-source relational database management system. Data such as web-server logs and other logs are stored locally in the gateway, enabling rapid append of data. Section Data Acquisition will explain the types of information being logged and the purpose of these data.

Controller

Bearicade's backend is the heart of Bearicade, driving and managing all components together. The controller is divided into three sub-layers in Bearicade's backend that together form the controller; this way, the complexity of the controller is reduced, reducing the load and issues that could occur in the system. The controller layers are:

- **Systems Controller**: This layer is what interacts with actual head nodes of the distributed system. It manages creating, reading, updating, deleting, and syncing local users to and from all nodes,
- **Terminal and File Manager Controller**: Terminal and File Manager described in section (Web-based Terminal and File Manager) uses this layer in order to initiate SSH login to different nodes, sending commands, and perform file manager actions.

- **Users Management Controller**: This layer has no major interaction with the nodes, as it's only used to store users' account information, credentials, activity, and actions. All of these data are stored in a separate database

On the ground of having multiple layers to the controller, interactions with the controller are managed by a single endpoint via a Representational State Transfer (REST) Application Program Interface (API). Especially with modularity in mind, REST API gives the ability to developers to design modular plugin for Bearicade, all with the least cost on performance on the gateway node.

By design, the controller is what manages all parts of the system, meaning that this layer is the most important layer of Bearicade in the sense of enhancing security. Therefore, it is crucial giving guidelines to this layer; guidelines will determine what the controller is able and not able to do. These guidelines are:

1. **Data validation**: part of quality control of the system, all data received from any user is validated against a set of requirement defined for each action. This includes the data type and data filtering.

2. **Predefined actions**: set of actions are predefined for the controller. The controller is not able to do additional actions unless implemented by administrators.

3. **Minimal User Input**: inputs from users are kept to a minimum to ensure the ease of data validation and reduce risk.

4. **Action Logs**: all activities that a user performs is logged automatically, as shown in table (3.3) logs include the what, who, and when data from an activity no matter what the outcomes are.

5. **Linux Permissions**: Bearicade Controller runs on the gateway system as a normal user with privileged access to necessary actions/commands only.

Web Portal

Bearicade is designed to rely the most of it's functionality on the backend, meaning multiple web portals could be developed on a single backend. Nevertheless, a web portal (frontend) is the only way users are interacting with the systems. Therefore there are multiple aspects that had to be considered in the design process for this project, as part of the functional and non-functional requirements for web portal. The frontend is where most of the requirements are implemented.

- Usability: part of the non-functional requirement for an HPC web portal as explained in section (Usability)
- Job and Data Management: most operative function for an HPC web portal, discussed in section (Job Management and Data Management)
- Performance: Bearicade does not impact the performance of HPC systems as it lies in a separate gateway node.

Keeping the requirements in mind, the web portal had to be designed accordingly.

Deployement of Bearicade

Deployement Orchestration

To keep consistency in the software architecture of the middleware described in section (Middleware), the deployment process followed a similar orchestration method used in (Higgins et al., 2018) by developing an ansible playbook. Ansible as an orchestration tool is a great choice for HPC since it allows access to the nodes in a cluster by using SSH with the default Public Key Infrastructure. This allows the tool to work without having third-party agents to be installed on the nodes nor creating a secondary Public Key Infrastructure key sets for the purpose of establishing a connection. Also, the configuration procedure is divided into roles which are stored in a YAML file on the system, allowing an administrator to easily toggle tasks, tweak the configuration, and define nodes by copying and modifying the

configuration file without the need for an external database, or central providers which is exactly what is needed for a geographically distributed systems such as grids.

Containerized vs Bare Metal Environment

Running the Bearicade stack or any software will require a bare metal server to run on, a containerization solution such as docker on top of the bare metal server could be used to isolate the software environment. This allows the software to be in a separate environment which tends to have a better security as it separates the hosts files from the container files. Containerized solutions could be an enhancement to a system when it is combined with an orchestration tool as it would allow the reconfiguration of the software without requiring to re-install Operating System on the host's system, without compatibility being an issue as it shares the hosts operating system. In regards to performance, containerization usually tends to add an additional layer to a system, such as a docker engine in the case of docker. In HPC systems, both performance and isolation are required (Xavier et al., 2013), and for this reason, the Bearicade stack could be installed on a separate server node that could act as the gateway node. This should not affect the performance of the actual HPC compute nodes at the same time allowing the isolation needed for better security practices and the ease of reconfiguration. Bearicade stack can be installed on a bare-metal machine without the use of containerization software. Also, it can be deployed on multiple containers to allow for better continuous development.

Bearicade Stack Deployement

Bearicade stack consists of 3 Ansible playbooks, each with its own roles. Playbooks allow administrators to run a fully configured and user-ready Bearicade stack

```
1 nodes:
2   - name: panda
3     ip: 167.71.138.98
4     ssh_port: 22
5     gateway: true
6     headnode: true
```

```
7  - name: polar
8    ip: 172.12.5.200
9    ssh_port: 22
10   gateway: false
11   headnode: true
12   workers:
13     - 172.12.5.201
14     - 172.12.5.202
15     - 172.12.5.203
```

Code 3.1 API for listing users

Table 3.1 Playbook roles

bear_install	**bear_add_user**	**bear_reinstall**
validation	validation	validation
mysql	bear/add_user	remove
php		**bear_install**
bear/bearify		
caddy		
traefik		
bear/post_bearify		

3.2.5 Data Acquisition

As mention in section 3.2.4, Bearicade is a data-driven framework, all of Bearicade's functionalities are made possible by the data it collects. Data is what makes the identification of a wide variety of attacks and the prevention of such attacks possible on a system. Thus, digital data collection is an important aspect of security for many organizations (Cobb et al., 2018).

As shown in figure 3.3, from an HPC user's perspective regarding granting access to the system, in order for the request made by the user attempting access to be approved, the user must provide multiple sets of data. The first set is the **User Identification Data** (UID); this data is stored as plain text in the database. It would be available for all administrators to preview at any point in time. User's Identification Data is as follows:

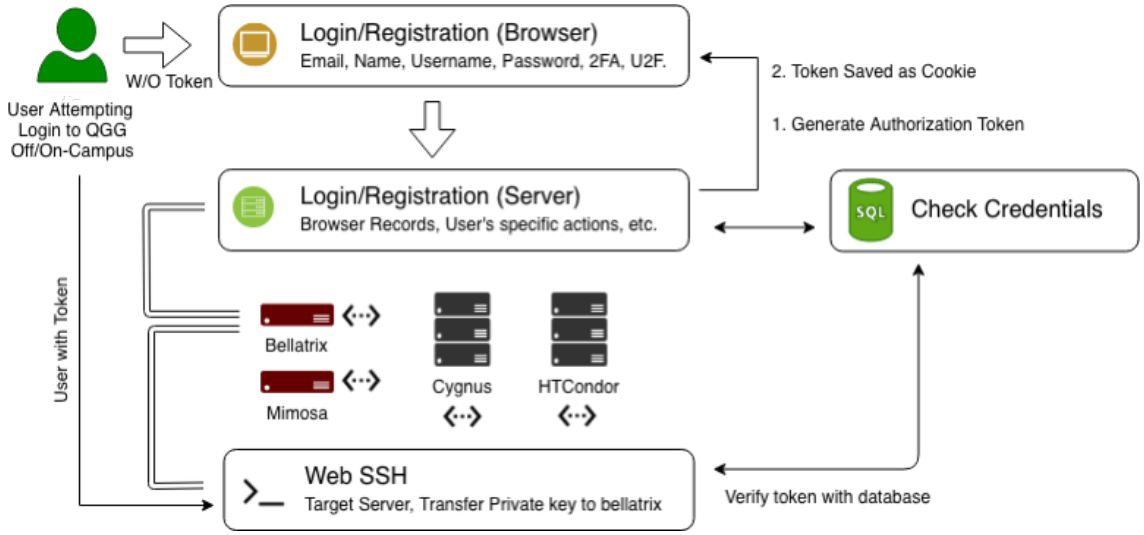

Fig. 3.3 HPC Systems Architecture in the University of Huddersfield from a user's perspective

- **Email Address**: In the registration phase, the vital data could be used to decide if the domain used in the registration to be allowed or not. For example ('*.ac.uk' could be used to only allow/deny International academic institution). Also, it is used to contact the user at any time. Depending on the scenario, it may be used as a multi-factor authentication if needed.

- **Name**: only used for reference; also, it could be used as part of the verification process if needed.

- **Username**: similar to name, with the addition of being used part of the student verification process (e.g. Student/Researcher/Staff ID).

- **Identification Number**: randomly generated and added to the user database, making it effortless for the system to manage users.

The second set of data are the **User Secret Data** (USD), the vital information that belongs to the user. The majority of these data are hidden from administrators.

- **Two-factor authentication secret**: This secret is randomly generated by the system using a Cryptographically secure pseudorandom number generator (CSPRNG). during

the registration process and is presented to the user to be saved or scanned via a Quick Response code (QR).

- **Password**: Taken from the user and checked on client-side and server-side for strength requirements predefined. The Password is hashed via bcrypt with an expensive key setup phase. Also, a 256-bit of random salt added to every hash, making the data is resilient to data breaches and rainbow tables.

The third set of data are **User Classification Data** (UCD). These data are collected by the system from the user's connection such as:

1. **Devices**: from every device the following data could collect:
    - System Languages.
    - System fonts.
    - Platform such as operating system, version.
    - Web programs such as Java.
    - Local system time.
2. **Browsers**, play a vital role in data collection, from the browser many values could be gathered, to help improving the system some of these data are:
    - Same domain Cookies (e.g. as the system domain is ***qgg.hud.ac.uk*** it is part of ***hud.ac.uk***, meaning all cookies set to ****.hud.ac.uk*** would be gathered.
    - Browser version and extension.
    - Navigation such as browser history and Referrer.
    - Screen resolution.
3. **Internet connection** is really important in our case since the system could collect Internet Service Provider (ISP) details, Internet protocol address and sometimes internal IP address. From these details we could identify:
    - Virtual Private Network is used (VPN), The Onion Router (TOR), Crawlers.

- IP Threat level (from popular databases).
- Country, and City (accuracy depending on ISP).
- Proxies used.
- Timezone (to be compared with browser and device).

The final set of data is the **User Behaviour Data** (UBD); these data summarizes how the user is interacting with Bearicade.

There are two components in this set, the first being the data collected from the users' interaction with the website via hardware devices such as keyboard or mouse. Collection of this information has to be done via the client-side, therefore, a Javascript library (Al-Jody, 2019) was implemented into Bearicade that would allow the collection of the behavioural information shown in table 3.2. Behavioural data is then passed to the backend server for validation and storage.

Table 3.2 Example of the behavioural data collected from Bearicade users

Data	Description
userInfo	record browser/device details
clicks	track mouse clicks
mouseMovement	track mouse movement
mouseMovementInterval	time between tracking mouse movements
mouseScroll	track mouse scroll
timeCount	track time

The second component is the user's activity which is collected from the actions the user performs from the web-based terminal and file manager, and if the user is an administrator, more data are collected such as create, read, update and delete (CRUD) functions on the system's data (i.e. users, API). An example of this information is shown in table 3.3.

Table 3.3 Action data collected from Bearicade users

ID	date	component	action
1	2020-01-17 11:27:58	System	Created User:"u1762746"
1	2020-02-03 12:08:30	User	Deleted:"106"
1	2020-03-01 18:13:11	Terminal	Command:"ls -la". Sent to: "cygnus" [Success]
1	2020-04-01 18:13:06	File Manager	Command:"ls" [Success]

More data are collected from unauthenticated users to help secure Bearicade from anonymous users. The information is collected when an anonymous user access Bearicade and attempts to authenticate. Attempts data includes Internet Protocol (IP) address of suspected users, along with the total of attempts and expiration date of a ban which is configurable within Bearicade.

Another set of information that is collected by Bearicade is the data from the web-server, and it holds records of requests made to any endpoint of Bearicade; such records holds data shown in table 3.4

Table 3.4 Example of Bearicade's web-server data log

Data Type	Information	Example
Request	Method	GET, POST
	Protocol	HTTP/2.0
	Scheme	HTTPS, HTTP
	Host, path, escaped queries, fragments	qgg.hud.ac.uk, /access ?action, #top
	Unix timestamp	
	Size	
	Status	200, 404, 301
	Latency	
User Identification	Man In The Middle detection	likely, unlikely
	Referrer	
	Cookie authentication token	
	Remote IP address and port	
	Browser user agent (OS, browser versions)	

3.2.6 Application Programming Interface (API)

The system uses an Application Program Interface (API) which is a software intermediary that contains predefined actions and dictates how the software's components should interact with the servers. API is needed so that it could authenticate the different users' interactions with the back-end servers, maintaining privacy and authority of the data.

When a client loads the web page, depending on the user's role/permission, the system renders all elements required for the selected role. When all the elements have loaded, the system multiple POST requests to the API in order to get all the data required to place for

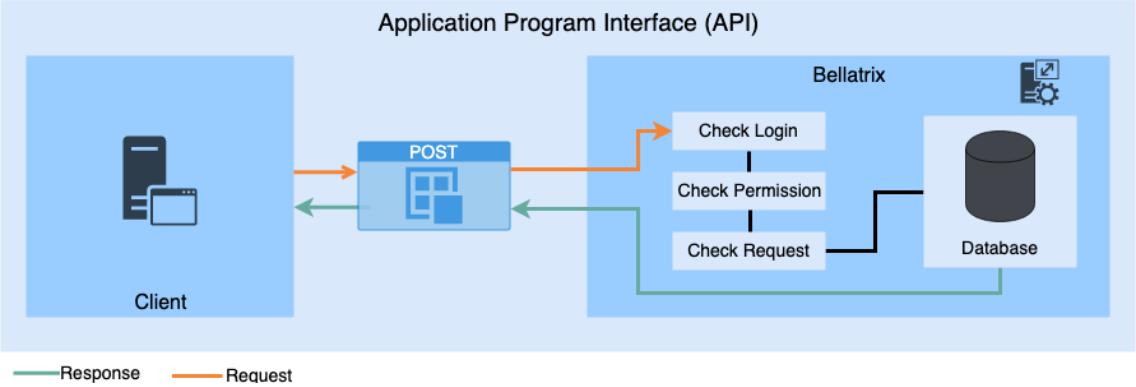

Fig. 3.4 Application Program Interface (API)

the elements. At this point, all the steps above are done on the client-side except for the page data. From now on, the API receives the request and starts processing it. As shown in figure 3.4, the API server checks for the users' Login token and tries to verify it in order to get the following variables:

- If user's token is active and logged in without a change in the User Classification Data (UCD).
- If the user has sufficient permission to run the requested action.
- If the user's request is valid and requiring an existing function from the API.

After all of the variables above are gathered and depending on the output, the API processes the request with the Database and returns a response to the client, which might contain valid output or an error for the user interface to process.

Below is an example of a request made to the API end-point, with no token, cookie, or action provided

```
1 > POST /controller/api.php HTTP/1.1
2 > Host: qgg.hud.ac.uk
3 > User-Agent: insomnia/6.5.4
4 > Accept: */*
5 > Content-Length: 0
```

Code 3.2 Request Header

```
< HTTP/1.1 200 OK
< Content-Type: application/json
< Server: Caddy
< X-Powered-By: PHP/7.2.14
< Date: Mon, 10 Jun 2019 21:42:34 GMT
< Content-Length: 30
```

Code 3.3 Response Header

```
{
  "error": "Permission Denied"
}
```

Code 3.4 Response Body (JSON)

From the example above, the server checks for the users provided details and returns an error: *permission denied*.

If the same request was sent again with modified header values, API will make sure to produce the required response. The request has an added query of ***a*** and value of ***failsCount*** in-order to return the number of failed logins for the system.

The Request header will be as follows:

```
> POST /controller/api.php?a=failsCount HTTP/1.1
> Host: qgg.hud.ac.uk
> User-Agent: insomnia/6.5.4
> cookie: authID=7b8f23a657ba8723ce6293257fcd9a80c3a40966; config=null
> Accept: */*
> Content-Length: 0
```

Code 3.5 Request Header

```
{
  "error": false,
  "isAdmin": true,
  "id": "1",
  "group": [
    {
```

```
      "name": "Administrator",
      "id": "1"
    }
  ],
  "failsCount": "515"
}
```

Code 3.6 Request Body

API Actions

As stated, the API has abundant functions to pull all required data for all software components. The API checks for permissions and the user's role before attempting to execute an action. Some actions that require extra permissions would be stated in the action declaration before attempting to query the data, for example:

```
//list all users action
if($isAdmin){
  $res['users'] = $conn->query("SELECT * FROM users")->fetchAll(PDO::
  FETCH_ASSOC);
}
else{
  $res = array('error' => 'Permission Denied');
}
```

Code 3.7 API for listing users

Example list of available actions for Users and Groups control and its functionality:

- **listUsers**: List all system users from all roles.
- **deleteUser**: Deletes a system user from the database without deleting local systems account or data.
- **toggleUserActive**: Enable or disable user's account temporarily.

- **listGroups**: List all system groups.
- **createGroup**: Create a group (different from local system groups).
- **viewUserBrowserRecords**: All devices and browsers a specific user has used to interact with the system.
- **viewUserActivity**: List what the user is performing on the system or the web dashboard.
- **viewUserLoginRecords**: Every login from a user.
- **viewUser**: List User Identification Data.

API Keys

The user requesting the action can be authenticated in two ways; the first is as explained before, being logged in and has a valid cookie containing the login data.

The second method is via API keys, this approach was implemented in order to give the system a modularity aspect where developers are able to incorporate an additional module that might be able to enhance the security or improve user experience.

```
1 MTIzNDU=.dab3734e2eaddcba5e4bb5a8234317e1
```

Code 3.8 API Key

Keys consists of two parts; the first is a base64 encoded ID of the user; the second part is the actual key (example 3.8). The key is generated by using an OpenSSL pseudo-random string of bytes consisting of 20 characters, following that MD5 hash is calculated of the random string, and the hash is then password hashed using the algorithm bcrypt.

API keys can be authenticated to access API the system only if no present cookie is provided.

User Authentication

Users accessing Bearicade's RESTful API end-point is a multi-process procedure. Figure 3.5, describes this procedure in details, and the processes go as follows as the users access the end-point:

1. The server checks for a present **authID** cookie; if a cookie is available, the server will check with the backend database to see if a valid session exist; if no valid session were found, then the system would deny the request.
2. If the server could not find a cookie, the server would check if the **HTTP_API_KEY** header is set. If the header is set, then thorough checks will be carried to confirm the key's validity:
    2.1. Key length is valid.
    2.2. Key containing two parts separated by a dot (.).
    2.3. Part of the key is an encoded base64 code.
    2.4. Part of the key returns a valid user ID.
3. if the session of the cookie active or API key is valid, the server will check if an action is supplied, which would then check if the user requesting the action is permitted to that particular action. If permitted, the server would process the action, returning the data requested. If not, the server would deny the request.

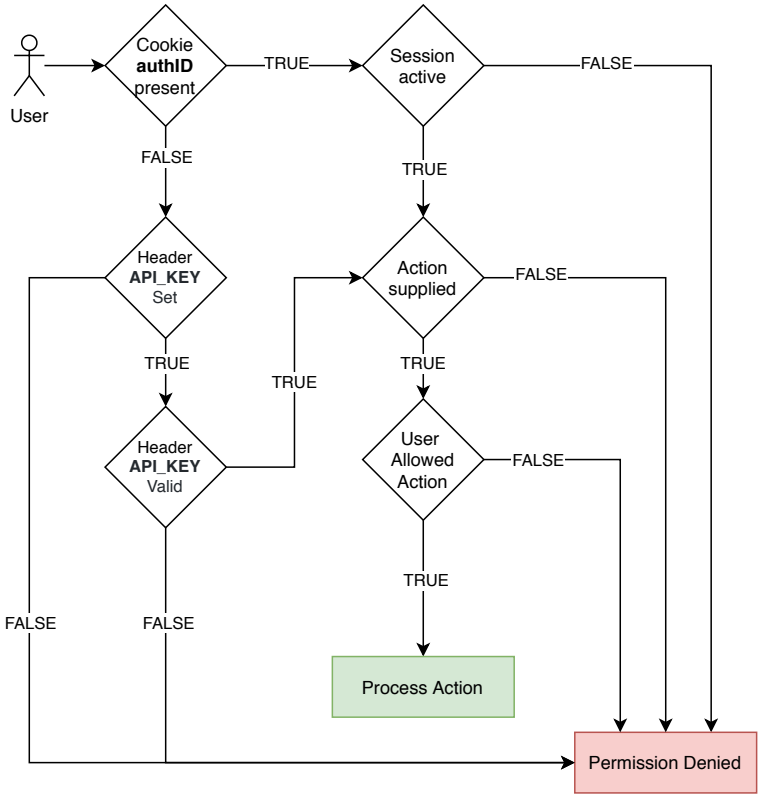

Fig. 3.5 Authenticating Users to Bearicade's RESTful API

3.2.7 Web-based Terminal and File Manager

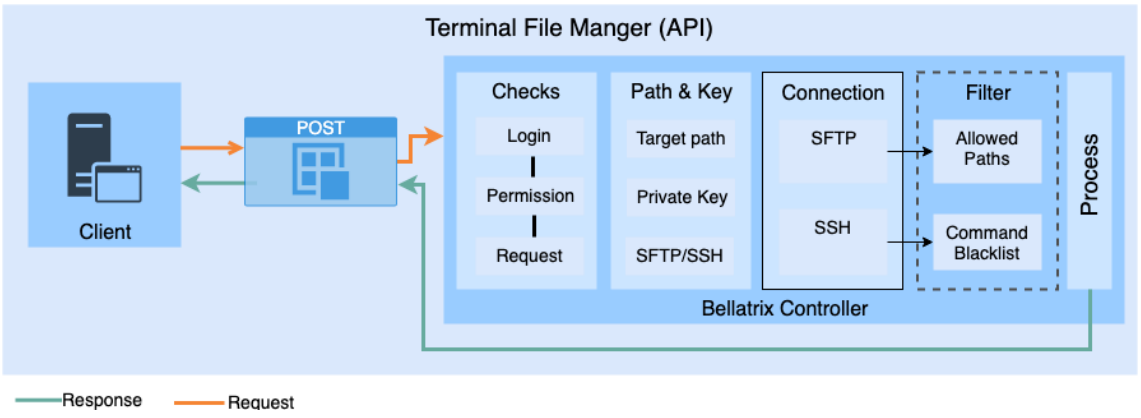

Fig. 3.6 Terminal and File manager API

Similar to the API used to establish a connection to the database, Terminal and File manager API follows the same architecture to allow users to access local files all from within the online portal.

As demonstrated in figure 3.6, following the checks made by the web-server for the session, permission and request. Following the initial checks, which would confirm the integrity of the user, the system checks for the target path attached to the request, ensuring the confidentiality of the data and the availability of the system are left intact.

Subsequently, the system loads a generated private key for the targeted user, depending on the requested action, an SFTP or SSH connection is created and is valid for one request only. This connection would still follow the systems policies and permissions and would log in as the user who commenced the request.

There are additional features in the system, which are smart filters that are able to deny a command providing that the command is blacklisted. For example, if the user runs a command in */home/user* folder such as:

```
rm -rf *
```

Code 3.9 Deleting files

This command would be allowed and passed to the system. However, if the user is running the following a slightly modified command in */home/user* folder such as:

```
cd /home/anotherUser/; rm -rf *
```

Code 3.10 Deleting another users files

This command would be denied by the filter, and the user attempting would be flagged, and their account would be deactivated. Clearly, even if the filter has failed and the command was passed to the system, user permissions of the system would deny it. The filter only gives an insight into the users' activity to systems administrators.

3.3 Security Implementations

According to William Cheswick and Steve Bellovin, "... any program, no matter how innocuous it seems, can harbour security holes ... We thus have a firm belief that everything

is guilty until proven innocent." (McGraw, 2006; Viega, 2002). A statement that has proven to be true by the number of cyberattacks that has happened targeting multiple institutions. In May 2020, a malicious group targeted academic data centres for CPU cryptocurrency mining purposes. The group has successfully hopped between compromised SSH accounts to get access to different HPC grids (EGI CSIRT, 2020). Many of the HPC centres have been affected by this cyberattack and this has caused a lengthy downtime in some cases. Fortunately for the University of Huddersfield's QueensGate Grid, Bearicade was put in place and remote SSH functionalities for users were disabled.

Throughout the years, there has been a number of published papers describing guidelines on building secure web applications. However, as technologies advances, cyber-attacks such as compromising HPC systems advances and becomes increasingly advantageous. Implementing a number of security mechanisms does not build an invulnerable system; instead, cybersecurity is a continuous process (Arbaugh, 2002). Software development is a challenging process, especially with limited resources (Fenton, 2014), adding to that building secure software is a time consuming and intense challenge (Abunadi and Alenezi, 2016).

3.3.1 The Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a nonprofit organization led by community security experts providing open-source projects, educational training conferences, and guidelines aiming to build secure software systems and securing the web. One of the most important projects by The Open Web Application Security Project is OWASP top-ten which focuses on developers and web application security, providing a broad list of the most critical security risks targeting web application. The latest version of this document is 2017; this data was gathered from specialized security firms, individual surveys, data span vulnerabilities from hundreds of organizations, and live applications and APIs (OWASP.org, 2017).

This section will focus on details of the top ten security risks explaining the risks that might have an effect on Bearicade, their impact, and prevention techniques. Table 3.5, shows the OWASP top ten list of security risks (A1-A10). Some of these risks arise when a particular

Table 3.5 OWASP top ten risks 2017

	Security Risk
A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging & Monitoring

technology is implemented in the web application. Therefore, some of the risks are not a potential problem for Bearicade, such as A4 XML External Entities (XXE) due to not having an XML processor.

Injection

A1 Injection is a top-ranked risk in the OWASP top ten for 2013 and 2017 issues. At any aspect of the application where there is data handling, there is a potential vector for injection. Since Bearicade relies on SQL database, there is a potential for SQL Injection vulnerabilities which could result in a data breach, corruption, unauthorized modification, and denial of service. Bearicade provides a safe REST API described in Application Programming Interface (API). The API provides a parameterized interface with server-side input validation in order to eliminate hostile data. Requests made to the API are only available to authenticated users, and all requests are logged for continuous monitoring.

Broken Authentication

A2 Broken Authentication is the 2nd top-ranked risk in the OWASP top ten for 2013 and 2017 issues, Broken Authentication is when an attacker has access to a list of login details combinations, or a breached datasets of username and passwords and the attacker is able to perform a brute force type of attack on the system by trying out combinations, this is

possible at any point of the authentication process. Bearicade's users are obliged to use a password with a specific strength that is then scrutinized on the backend using a pattern matching and minimum entropy calculation where a minimum score that is measured on the probability of guesses is allowed. Another prevention method implemented on Bearicade is a Time-based One-Time Password (TOTP) two-factor authentication (2FA) system. The users are not able to register on the system without 2FA. Also, there is a configurable limit on the number of tries for failed login attempts. Upon exceeding the limit, the attempting user will be banned by IP address. Further prevention could be done by disabling the user account that is being targeted.

Sensitive Data Exposure

A3 Sensitive Data Exposure is performed by an attacker by stealing data from the user's client such as a browser through man-in-the-middle (MITM) attacks or gathering plain-text data from a server through a data breach, or performing cryptographic decryption on data that was encrypted using a weak algorithm. Although MITM attacks usually target the client-side and most popular browsers and security software have implemented a method to protect the user against them, misconfigured backends could assist an attacker in successfully hijacking the client's communication with the server.

Table 3.6 Recent SSL/TLS Vulnerabilities and Bearicade's protection against exploitability

Exploits	Not Vulnerable
Heartbleed (CVE-2014-0160)	✓
CCS (CVE-2014-0224)	✓
Ticketbleed (CVE-2016-9244)	✓
Secure Renegotiation (CVE-2009-3555)	✓
Secure Client-Initiated Renegotiation (CVE-2011-1473)	✓
CRIME, TLS (CVE-2012-4929)	✓
BREACH (CVE-2013-3587)	✓
POODLE, SSL (CVE-2014-3566)	✓
SWEET32 (CVE-2016-2183 CVE-2016-6329)	✓
FREAK (CVE-2015-0204)	✓
DROWN (CVE-2016-0800 CVE-2016-0703)	✓
LOGJAM (CVE-2015-4000)	✓
BEAST (CVE-2011-3389)	✓
LUCKY13 (CVE-2013-0169)	✓
RC4 (CVE-2013-2566 CVE-2015-2808)	✓

In the last ten years, there has been a number of Common Vulnerabilities and Exposures (CVE) shown in table 3.6 that has been used in attacks such as MITM. These CVEs have targeted specific versions of SSL/TLS. In order to be well-protected against these CVEs, Bearicade is not supporting older versions of SSL/TLS and only offer a successful handshake with browsers supporting TLS 1.2 and TLS 1.3. This will cause outdated and insecure browsers with a handshake failure error. Moreover, Bearicade's web server uses 2048 bits SHA256 key with RSA and have explicitly defined supported cypher suites denying downgrade attacks and decryption of secure communication. Using a method described in (Durumeric et al., 2017) as a prevention technique, Bearicade's web server uses a plugin to help detect HTTPS interception by grading the user's client using tests such as (i) TLS Versions supported (ii) Cipher Suites provided (iii) Multiple Certificate Validation (iv) Known TLS Attacks. Bearicade will not authorise clients scoring a low grade, presenting them with a message, and the request will be logged.

Broken Access Control

A5 Broken Access Control are one of the primary exploitation that is performed by an attacker to trick the server into performing a request on behalf of other users without the

required permissions, and this could lead to a leak, unintended modification, or loss of data. A 2018 study has analysed 330 web applications from the educational, governmental, health, and private sectors. The analysis conducted has concluded that 39% of the web applications tested were vulnerable to Broken Access Control attacks caused by improper input validation, data disclosure, and insecure sessions (Hassan et al., 2018). As discussed, by design Bearicade does not trust user inputs from unauthorised users to administrative accounts; this is why user inputs go through input scrutinisation on both client and server-side. Also, when an authenticated user performs a request to Bearicade's API, the user does not explicitly provide a piece of identification information; instead, a cookie is passed along with every request, the scrutinisation cookie value is then checked for its validity by checking active sessions, then the user's device user agent and IP address are verified against the initiator's user agent and IP address. This process repeats in the backend server every time a request is initiated; this control mechanism is present throughout the application.

Header Configuration

Bearicade provides an API to third-party applications in order to build modular plugins for Bearicade, allowing the implementation of more advanced features and security enhancements. For this reason, Cross-Origin Resource Sharing (CORS) has been implemented into the API. CORS is a system that determines whether an application is allowed to access the responses from the server, overriding same-origin security policies that by default forbid cross-origin resource sharing. By transmitting HTTP headers in what is called a preflight request, CORS checks that the server is cognizant of the user's actions defined in the header.

```
1  header("Access-Control-Allow-Methods: GET");
2  header("Access-Control-Allow-Origin: dev1.qgg.hud.ac.uk");
3  header("Access-Control-Allow-Headers: API_KEY");
4  header("X-Frame-Options: deny");
5  header("X-XSS-Protection: 1; mode=block");
6  header("Content-Security-Policy: script-src https://*.qgg.hud.ac.uk")
7  header("Strict-Transport-Security: max-age=86400");
8  header("X-Content-Type-Options: nosniff");
```

```
9  header("Content-Type: application/json");
```

Code 3.11 Bearicade's Response Headers Confirguration

Bearicade defines a dynamic Allowed Access Control Methods, Origin, and Headers from the application's request depending on the user and the action performed. As shown in code 3.11 lines 1-3, the server will only allow a GET request made from *dev1.qgg.hud.ac.uk* with a supplied API_KEY header key. Line 4 X-Frame-Options deny the browser from rendering the content of the page in a frame element (e.g., frame, iframe, embed, or object); this is a protection procedure to protect the users from falling into click-jacking attacks by making sure the content is only accessible from the origin site. Line 5-6 is another attack prevention feature that commands supported browsers to block a request if reflected cross-site scripting (XSS) is detected. This feature is most useful for older browser complementing the newer Content Security Policy implementation in browsers to allow scripts to be loaded from the same domain. Line 7 Strict-Transport-Security ensures that the browser should only access the page via HTTPS. Lines 8-9 deny applications Multipurpose Internet Mail Extensions (MIME) type sniffing and make sure that the application process responds as a JSON document.

Security Misconfiguration and Using Components with Known Vulnerabilities

Improper set up in security-related configurations in web applications and web servers may result in opening up opportunities for attackers to exploit systems (Mendes et al., 2008). Also, a large number of Internet-connected devices are using vulnerable components as part of their system, allowing attackers to perform an automated large scale exploits (Cadariu et al., 2015). Both of Security Misconfiguration and Using Components with Known Vulnerabilities are security concerns that could be found in different layers of a system, including web servers, databases, frameworks, or libraries, among others.

From Bearicade's configuration shown in 3.7, Bearicade was designed on a minimal platform as suggested by OWASP (OWASP, 2017), consisting of 3 base components: (i) Caddy

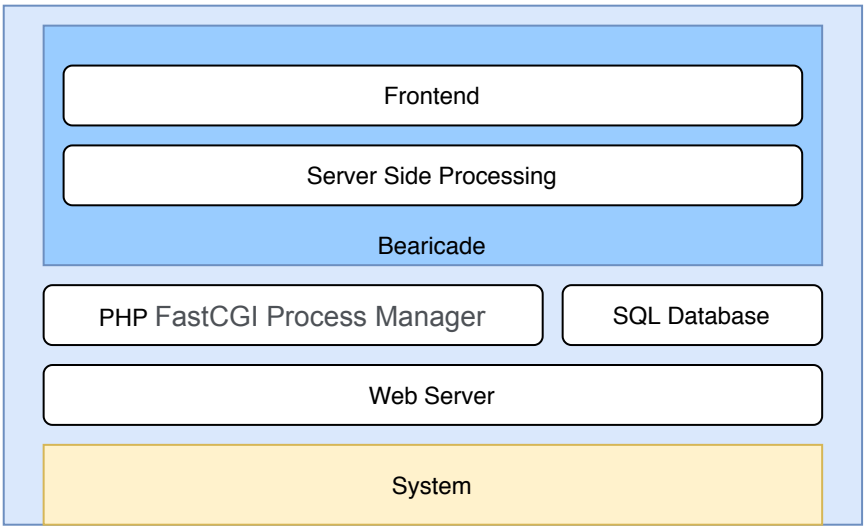

Fig. 3.7 Bearicade's Configuration

(Caddy, 2020) is used as the webserver of choice for bearicade because of it is minimalism and security features (Cuppens, 2016) (ii) MySQL Database (iii) PHP FastCGI Process Manager, all of these components are containerized by default to ensure supplemental isolation and customizable during Bearicade's installation. Secure configuration for each of these components is followed respectively from the documentation of each component.

On top of the three base components, there are two components that form the web application that is the server-side and the frontend layer. The server-side layer consists of PHP programs that are compiled on the server. All libraries used in this layer are installed via a dependency manager that defines the library and the version; this implementation is valid for continuously inventorying the versions and checking for active vulnerabilities related to these dependencies. The frontend consists of Javascript libraries, which are also continuously checked using version scanners.

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) enables attackers to compromise a web page viewed by other users by injecting client-side scripts. According to OWASP, This type of attack is found

in around two-thirds of all applications, endangering users by leading to stolen credentials, session hijacking, or delivering malware.

In addition to the security precautions described in section (Header Configuration), there are extra measurements to prevent XSS: (i) Bearicade uses Vue.js (Vue.js, 2020) as a progressive Javascript framework, and by design, Vue.js escape HTML content (ii) automated API to escape and sanitize content data sent and received (iii) performing scheduled system scan using the Security Content Automation Protocol (SCAP) standard to detect potential vulnerabilities.

Insufficient Logging & Monitoring

With all the security implementations put in place to protect against OWASP top ten security risks, attackers are still managing to intrude into the system by relying on insufficient logging, monitoring, and timely response from the system maintainers. A 2017 report made by Ponemon Institute (Ponemon Institute, 2017) shows that in 2016 data breach attacks took on average 191 days to be identified and an average of 66 days to be contained, enough time for attackers to cause harm to the system.

As described in section Data Acquisition, Bearicade collects a large amount of data from the interaction with the system, resulting in leverage of potential monitoring and alert mechanisms to be implemented in the system giving administrators a way of visualizing this information through the dashboard. Use-cases of such mechanisms will be discussed and proven further in the Artificial intelligence-based tools for secure access to HPC resources chapter.

3.3.2 User Authentication

Bearicade uses a modified open-sourced PHP authentication class (PHPAuth, 2020); the class was published in 2014 and currently has a large number of contributors, with undergoing

development, auditing, and bug fixes, making it more reliable to be used when compared to making an authentication class from scratch.

Email

Bearicade could be configured with a allow/deny-list to either accept or reject email addresses belonging to specific domain names. For example, the University of Huddersfield are able to accept email addresses with the **.hud.ac.uk* prefix in order to allow individuals from the institute. Moreover, there is an implementation to check an email address against a list of disposable email address domains.

Password

Cryptographic hashing is a one-way function that takes a value as an input and generates a fixed-length unique message digest, also known as a hash. Unlike traditional encryption functions, which is a two-way function, hashing cannot be reversed, making it impossible to generate the original input value (Ertaul et al., 2016).

According to a 2016 study, PBKDF2, Bcrypt, and Scrypt are hashing algorithms providing the best security (Ertaul et al., 2016). Bearicade uses Bcrypt to hash passwords. Since it was created in 1999 (Provos and Mazieres, 1999), Bcrypt is a memory hardening function, making it CPU and RAM intensive as well as GPU resilient by design (Metla et al., 2018).

Some cryptographic hashing functions are vulnerable to rainbow tables attack. In a case of a data leak, attackers would be able to compare the hashed values in the data and compare it with a list of hashed common passwords (rainbow table) until a match is found. In order to thwart such an attack, a unique generated value (salt) needs to be added to the input while hashing; this function is called salting. Bearicade authentication system pulls a unique 16-byte salt from /dev/urandom for each password hashed.

Brute-force attacks are when an attacker attempts to guess the password by trying a random combination of characters, or a database of commonly used passwords (dictionary attack), this type of attacks could happen online on a live system or offline in case of a data leak. In

order to prevent online attacks, Bearicade limits the number of login attempts to 5 attempts per user; exceeding the limit will result in banning the offender. When it comes to offline attacks, the current hashing function uses a "cost" parameter which is the power of two number (2^{cost}), stating the number of rounds to repeat the bcrypt function, resulting in an exponential increase in the time taken to calculate the hash.

During registration, the users are required to input a password that needs to meet a specific strength. Calculation of the strength made possible using pattern matching and minimum entropy calculation, inputted password are pre-checked on the client-side then post-confirmed on the server-side using zxcvbn which was developed by Dropbox (Wheeler, 2016).

Two-Factor Authentication

Two-Factor Authentication (2FA) is an essential step to authenticating users by defending systems against account compromise (Reese et al., 2019). Bearicade is implemented with Time-Based One-Time Password algorithm (TOTP) (Corp Pei Symantec J Rydell, 2011), during authentication the user is required to present a One-Time Password (OTP), OTP values are short-lived depending on the time factor. During registration, users are presented with a random secret along with a Quick Response (QR) code. Users then can add the secret/QR code to their prefered 2FA application. This process adds a verification to the user's authentication using a piece of hardware device they have, such as smart-phones. This procedure was achieved using the TwoFactorAuth PHP library (Janssen, 2020).

3.4 Dashboard Features

This section will explore the graphical user interface of Bearicade (dashboard), where the users can interact with the features implemented in the dashboard and access the clusters configured. There are currently two versions of the dashboard, the first being for the regular users giving them access to shell and file manager. The second version is for the

administrators, which provides the same features as regular users with added functionality to monitor and manage the system and its users.

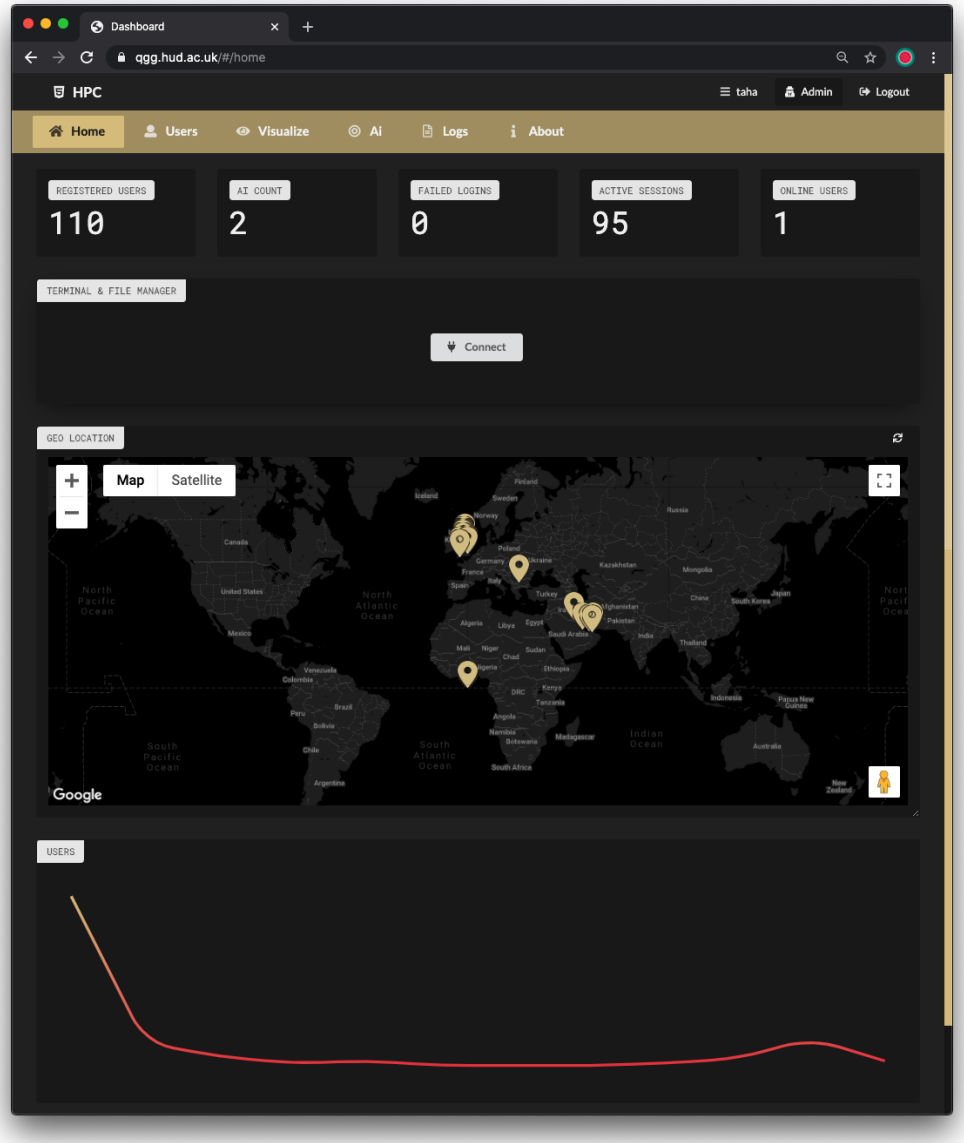

Fig. 3.8 Bearicade Dashboard: Administrator Main View

Figure 3.8 shows a screenshot of administrators' entry point on Bearicade's dashboard. On the main page, administrators are presented with (i) navigation bar (ii) basic statistics such as number of registered users, recently failed logins, current active sessions, and online users

(iii) terminal and file manager element (iv) geographical location of currently active sessions (v) graph showing new users activity.

3.4.1 Terminal and File manager

The terminal and file manager element is the key feature of Bearicade's dashboard presented to both administrators and standard users. Upon initiating a connection by pressing the connect button, the element will display the file manager GUI as shown in figure 3.9. The GUI has three key features that are linked using icon illustrated buttons:

- Terminal and File manager toggle button, the button will allow the user to switch between the terminal and file manager.
- Upload

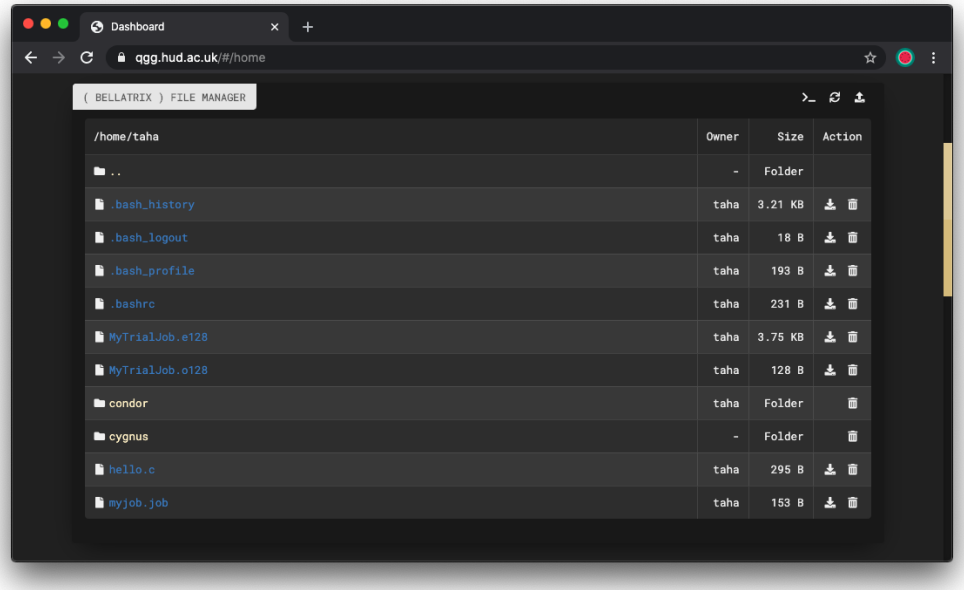

Fig. 3.9 Bearicade Dashboard: File Manager

File Editor

Using the file manager, the users can view the files on the local storage server along with metadata such as file size and the ownership of the file. The user can click a file in order to edit, and the element will show a file editor as shown in figure 3.10 and the user can make changes and save or discard. The user could also choose one of the actions available for files, such as download or delete the file. Clicking a folder would result in changing the directory of the file manager.

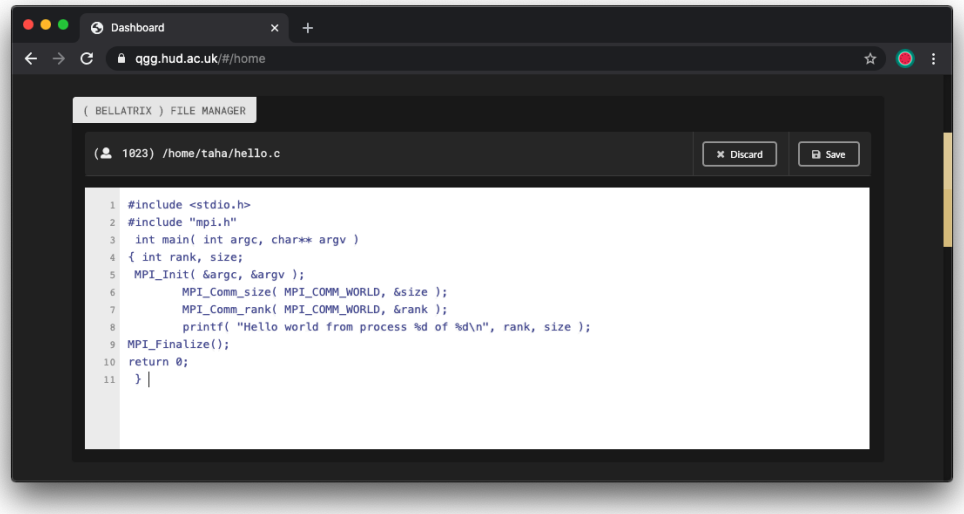

Fig. 3.10 Bearicade Dashboard: File Editor

File Upload

Users are able to upload files conveniently from within the file manager using the upload button, as demonstrated in figure 3.11 users are able to drag and drop or choose files wishing to upload to the local cluster storage.

Fig. 3.11 Bearicade Dashboard: File Upload

Server Selector

In the case of the University of Huddersfield's QueensGate Grid, all the clusters use a single network shared storage nodes, meaning that the file-manager is the same for the users on all clusters. However, for the terminal, the users should be allowed to switch between different clusters in order to submit jobs on different clusters on the grid. This could be done using the server selector, which is integrated on the File-manager/terminal element on the dashboard as shown in figure 3.12. Administrators are able to allow and deny users from accessing a particular cluster within the configuration.

Fig. 3.12 Bearicade Dashboard: Server Selector

3.4.2 Account Information and Customization

From within the dashboard, the users are able to view and update limited information regarding their account via the sidebar as shown in figure 3.13; this information includes Full name, Email address, and the 2FA Secret along with the QR code. Providing the user with visual customization, there is a toggle implemented in Bearicade's dashboard for the user to be able to switch between dark and light theme as shown in figure 3.14. The dashboard is set to the dark theme by default; however, user's preference will be saved on their browser via a cookie.

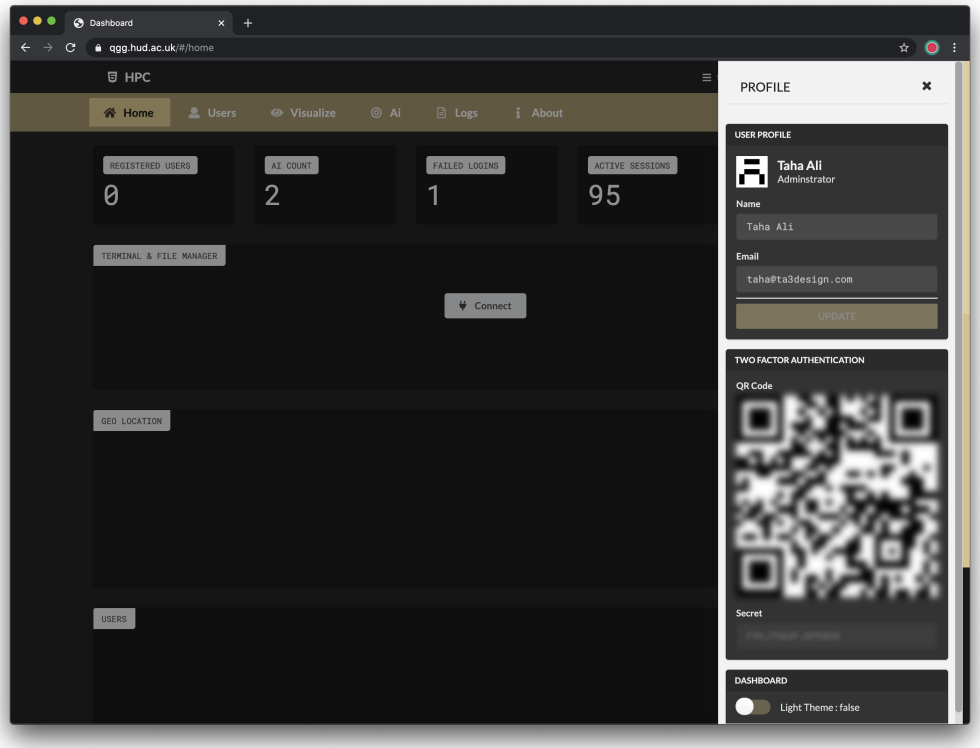

Fig. 3.13 Bearicade Dashboard: User Details

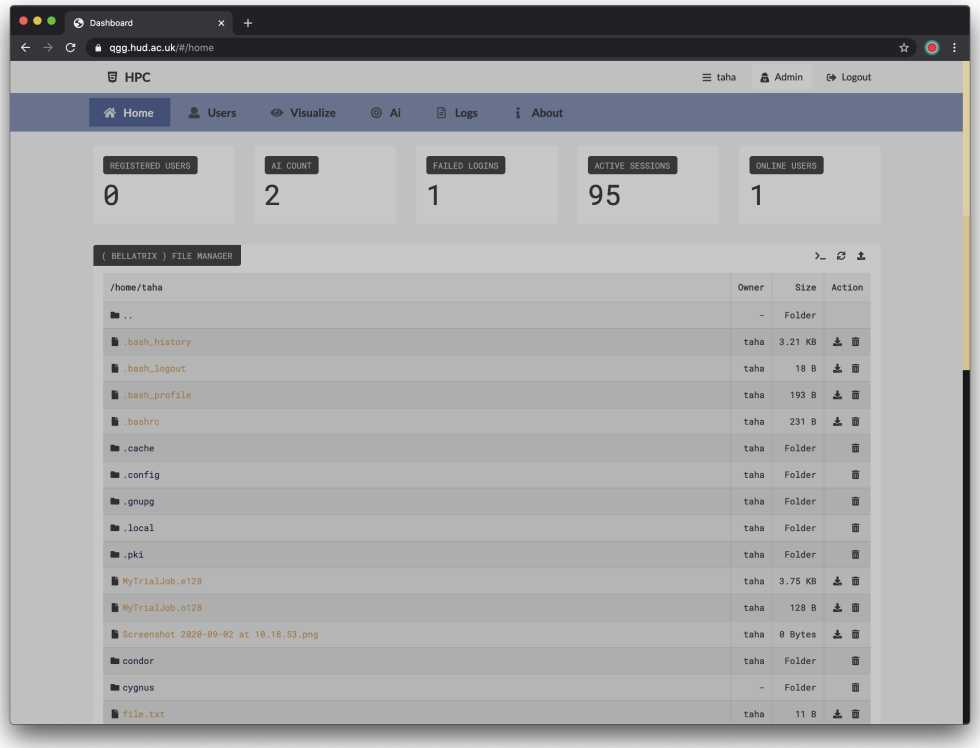

Fig. 3.14 Bearicade Dashboard: Light Theme

3.4.3 Users Management

Bearicade's Users Management is dedicated to the administrators. This section will cover most of the Bearicades user interface's functionalities part of the users' management. The part of the user interface shown in figure 3.15 demonstrates users' main page. The page has a list of all users registered on Bearicade. Through the list, administrators are able to view the users' full name, username, and email, administrators, are also able to search via this information.

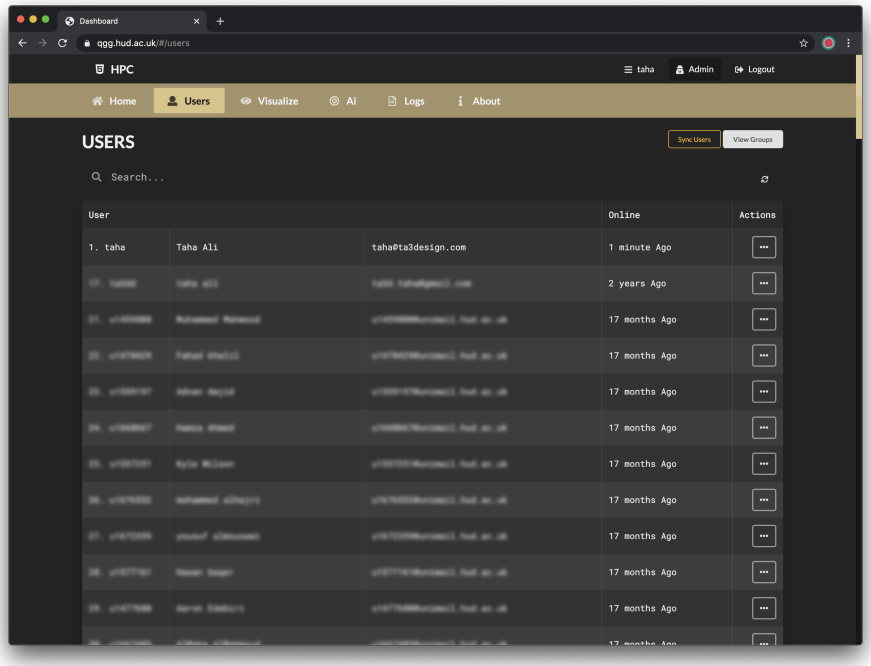

Fig. 3.15 Bearicade Dashboard: Users List

User Details and Actions

Through the users lists, administrators are able to view and control different actions per user, as shown in figure 3.16, details about the user are available such as (i) Identification number (ii) Time since last active (iii) Groups (iv) Joining Date (v) Time-based one time password secret (vi) futher details will be discussed in the following sections. In addition, actions available are (i) Acivating/de-activating (Lock/Unlock) accounts (ii) Removing System User from systems with or without the user's system files (iii) Deleting Bearicade accounts.

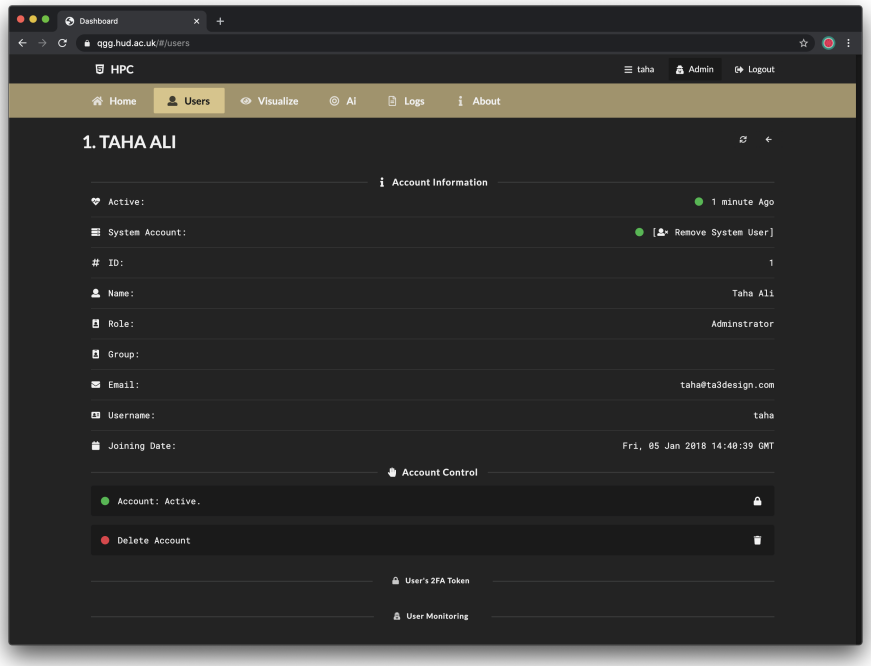

Fig. 3.16 Bearicade Dashboard: Users Details

User Activity and Monitoring

Users activity is essential for continuously track and prevent potential security risk along with diagnosing possible issues. Bearicade provides a range of data visualization elements. As shown in figure 3.17, these elements are divided into 3 sections (i) Browser Records (ii) Interaction Activities (iii) Login History. The data in each of these elements are described in details in section Data Acquisition.

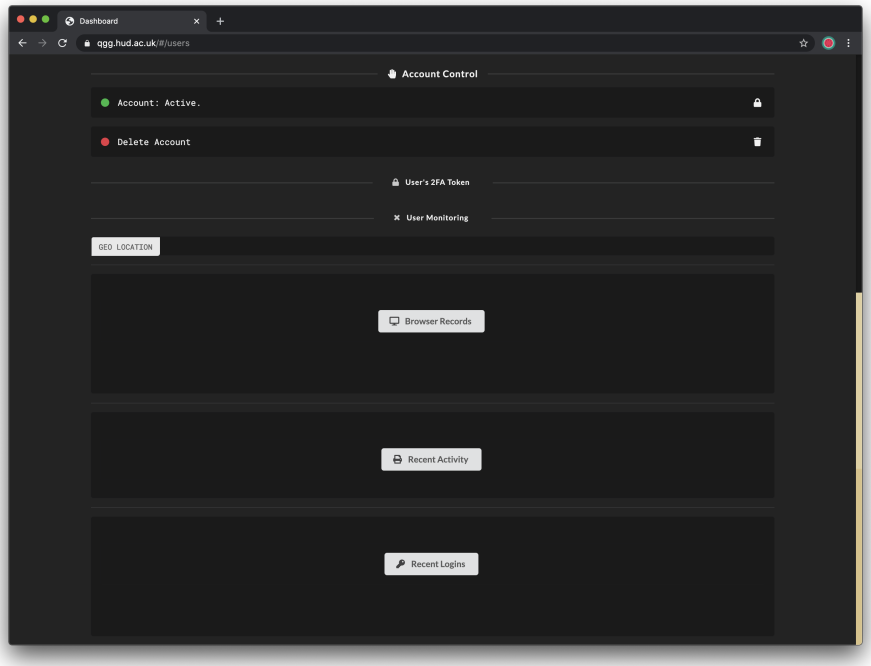

Fig. 3.17 Bearicade Dashboard: Users Monitoring

Subsequent to each successful login, Bearicade records information regarding the user's browser and device as shown in figure 3.22 such as: (i) IP address (ii) Timestamp (iii) Frequency of the unique record. (iv) device fingerprint (v) cookies (vi) languages (vii) browser and user-agent (viii) platform (ix) screen-size (x) referrer. Along with this information, Bearicade logs the number of failed login attempts as shown in figure 3.20.

As shown in figure 3.19, the last element of the activity elements provides information about the user's interaction with the system, such as the use of the terminal and file-manager. This information include the timestamp of the activity, type of the activity, action attempted, system targeted, and the result of the activity.

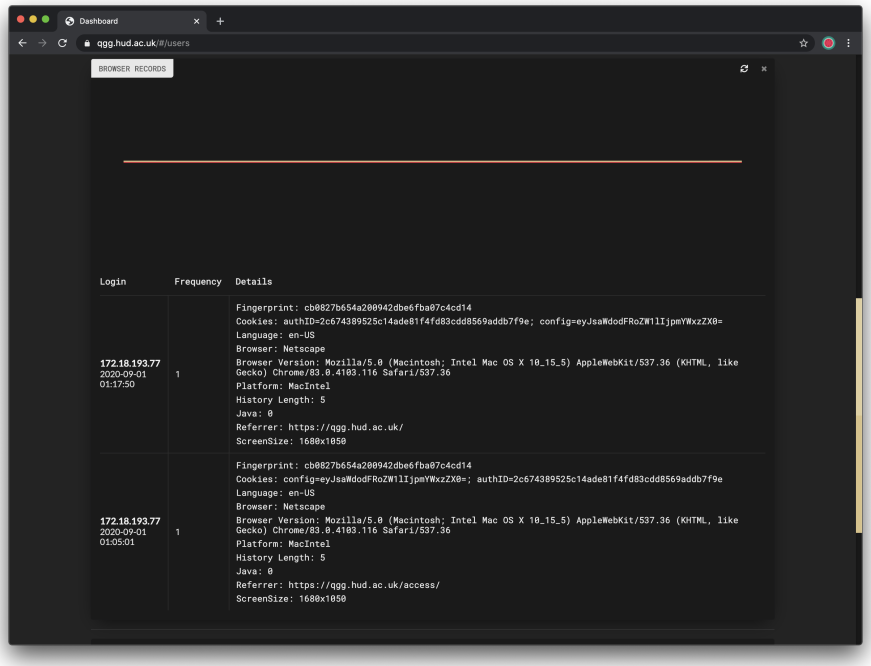

Fig. 3.18 Bearicade Dashboard: Users Browser Records

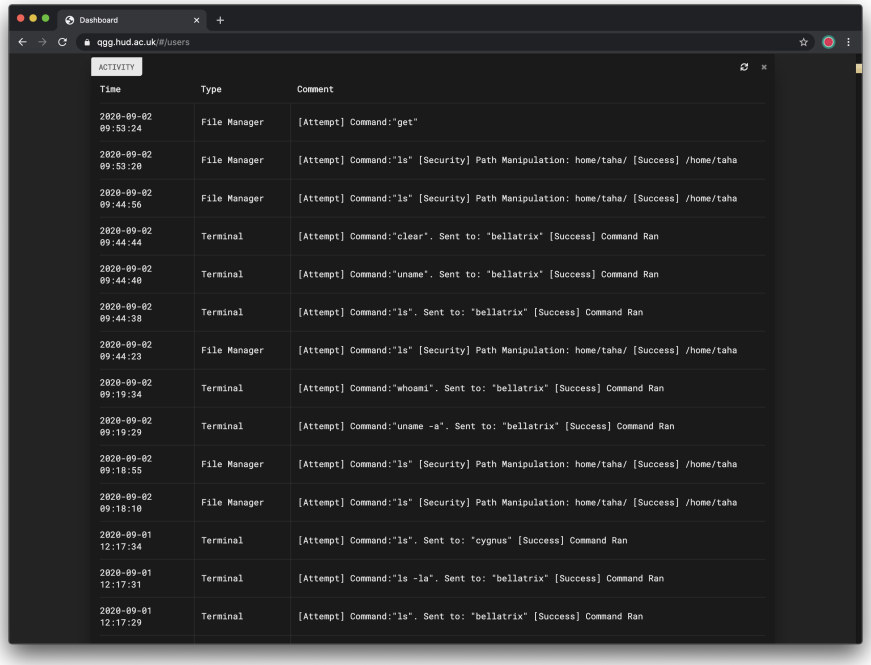

Fig. 3.19 Bearicade Dashboard: Users Activity

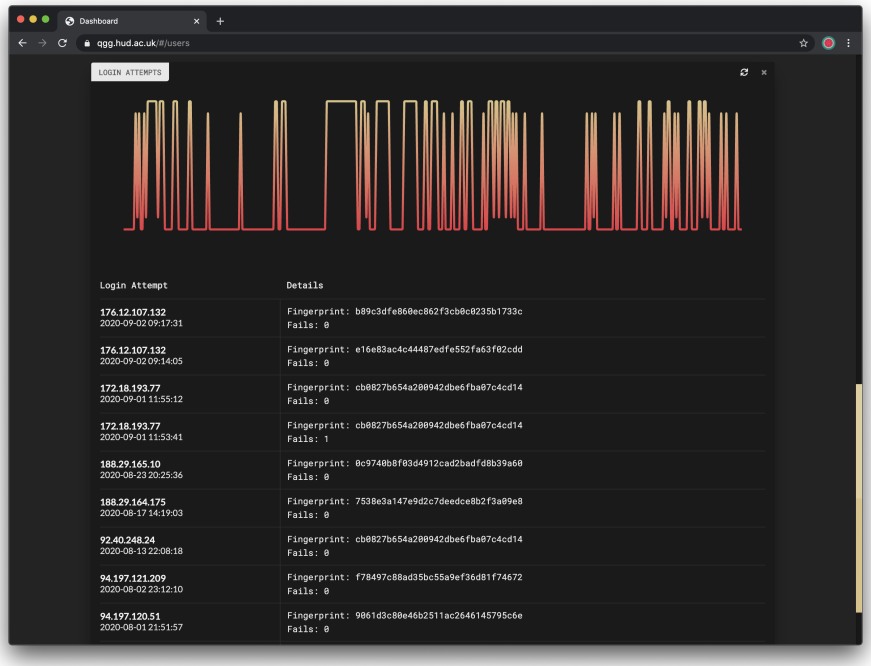

Fig. 3.20 Bearicade Dashboard: User Login Attempts

3.4.4 Settings

The settings section provides the possibility of changing the system and Bearicade's settings from the browser; these settings are divided into six sections:

1. Site: availability to change site name, base URL, administrator email, and other basic settings.
2. Security: settings provide the ability to quickly modifying security settings such as attach mitigation time and attempts before ban.
3. SMTP: Secure Mail Transport Protocol (e-Mailing) settings.
4. Cookies: Modifying cookies parameters stored in users' browsers.
5. Logs: logs path to be collected and visualised on Bearicade.
6. Server: Web server configuration settings.

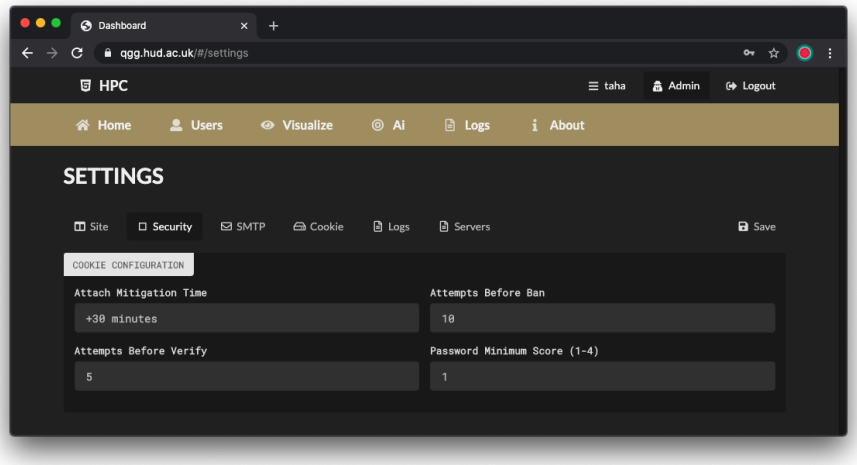

Fig. 3.21 Bearicade Dashboard: Security Settings

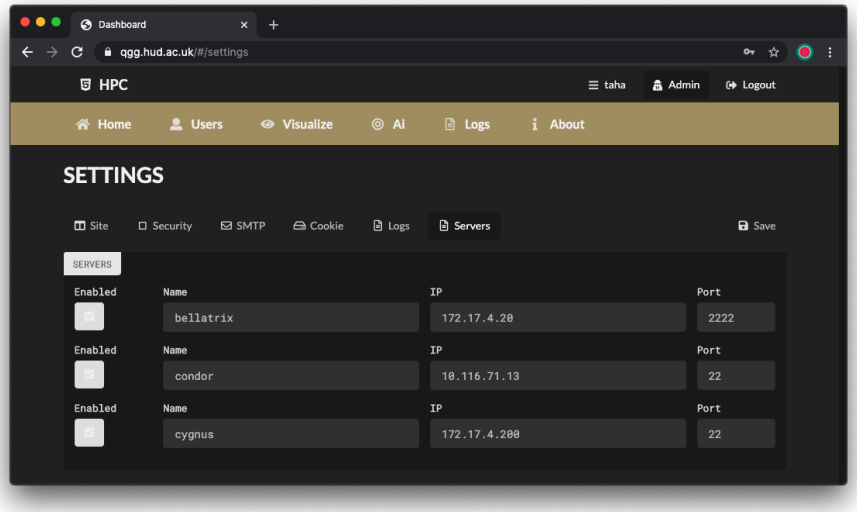

Fig. 3.22 Bearicade Dashboard: Servers Settings

3.5 Summary

Bearicade is a users and security management gateway framework for distributed systems, permitting users to remotely access multiple nodes on/off-premise, providing a web terminal and file-manager implementation. Built on a restful API, Bearicade allows administrators to visualise data. Most importantly, Bearicade enables machine learning implementation to

help to enhance security from the data collected.

In order to evaluate the performance of the Bearicade framework in typical scenarios relevant to secure access to the HPC systems, a number of experiments were conducted on the QGG grid HPC resources. They will be presented in the upcoming chapters.

Chapter 4

Deployment of Bearicade

4.1 HPC Grid

The University of Huddersfield's HPC services campus grid is known as QueensGate Grid. Students, researchers, and staff make up a diverse group of users on the grid. Users come from various departments such as Sciences, Engineering, and Business, which means the grid must be suitable for a wide range of use cases, including DL-Poly, OpenFoam, Ansys, Matlab, Maxwell, Fluent, and COMSOL, to name a few.

4.1.1 Systems Distribution

At the time of writing, the University of Huddersfield's QueensGate Grid HPC systems consists of 4 Clusters, all of which are geographically apart.

At the University of Huddersfield, there are three main High-Performance Computing systems distributed around the campus. As shown in figure 4.1, there are two central locations for HPC systems. Data Centre is the first one, which holds:

- Cygnus, which is a cluster, consists of 9 nodes, each with 24 cores Intel Xeon X5690 CPU clocked at 3.47GHz, and 200 Gigabyte of memory.

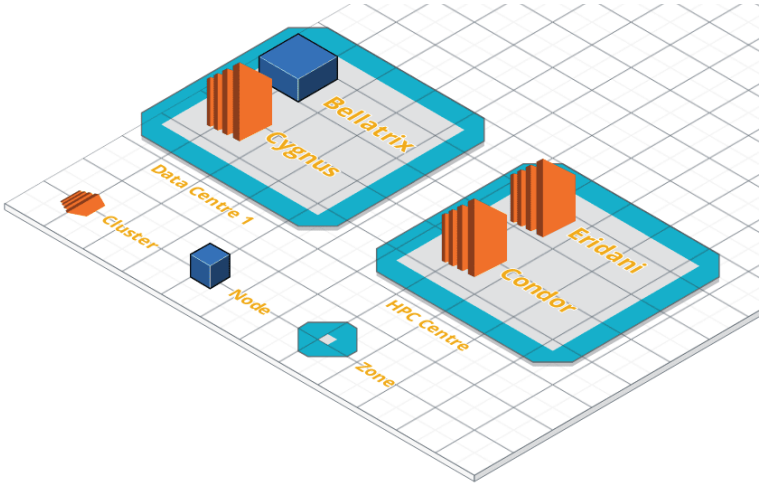

Fig. 4.1 HPC Systems Distribution at the University of Huddersfield

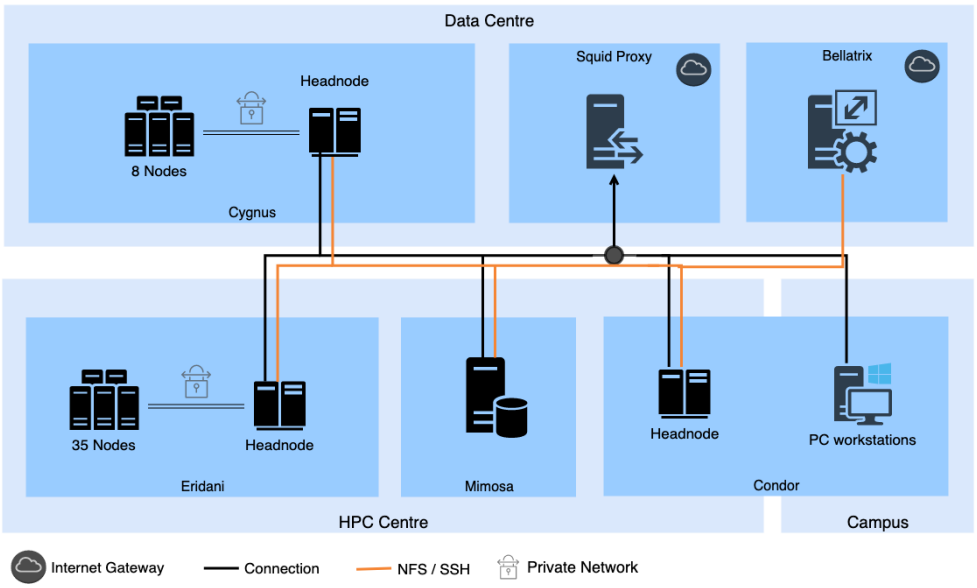

Fig. 4.2 HPC Systems Architecture in the University of Huddersfield QGG

- Bellatrix, which is the chief node for controlling and managing all the connections to all clusters and nodes. It has a major part in the proposed framework.
- Orion is a cluster used for scientific research; it was not used to gather data for the experiments.

The other location in the QGG campus grid - HPC centre houses:

- Eridani, a 35 nodes cluster each node has 8 cores Intel i5 clocked at 3.2GHz, and 8 Gigabyte of memory.
- Condor, which is a high-throughput computing software framework for Windows and it runs on over 300 windows computers distributed around campus. The head node, however, is located in the HPC Centre.

Each one of the head nodes for each of the clusters is connected together via the University's network (HPC VLAN and DC1 VLAN). The clusters' nodes are connected via a private network without being accessible via the University's VLAN.

4.1.2 Access Software at the University of Huddersfield

Predecessor Access Software

Like many of the HPC systems around the world, the previous access setup at the University of Huddersfield in order to access the various systems of the grid was a Linux based node that is setup with Secure Shell (SSH). The node was set up to be accessed externally by exposing the SSH port externally. By logging in to SSH, the users could then access all head nodes from the terminal. For a new user to be registered in the system, system administrators have to create a local user and generate a Public Key Infrastructure (PKI) set of keys and the local users along with the home folder is then synced with the rest of the nodes in the grid. The newly registered user should then receive their private key via email.

Current Access Software

At the University of Huddersfield, and since 2018, Bearicade has been the main access software for QueensGate Grid. Bearicade allows users to access all clusters at QueensGate Grid described in Systems Distribution remotely.

4.1.3 HPC User Access and Privileges

The predecessor access software (SSH) at the University of Huddersfield has only allowed two types of users to access the system, a user and an administrator. This method did not provide enough security metrics such as data about the users' behaviour, data visualization, and alerts.

Bearicade has introduced new and more customizable privileges. This section will introduce three categories of users in Bearicade at the University of Huddersfield.

Researchers and Students Users

For every academic year, the University of Huddersfield's researchers and students request access to use QGG. Approved users will have access to the head node of each of the clusters they are permitted to. From within the dashboard, the students have the ability to access the basic functionality of Bearicade, such as (i) File-manager (ii) Terminal (iii) List of clusters Enrolled masters and undergraduate students to the Parallel Computer Architectures Cluster, Grids and Cloud Computing module are assigned weekly activities that they are required to perform on the QueensGate Grid clusters. In order to achieve these tasks, the users are required to login to Bearicade and interact with the systems from within the dashboard. This means that the users will log in to Bearicade multiple times every week throughout the academic year from unique devices, different times and perform different actions.

Other researchers such as Ph.D/Masters students and academic staff can have access to the system to submit jobs and perform other tasks on the system.

These users are the main source for the data collected by Bearicade. The data could then be used to train machine learning models in order to have an understanding of the patterns of the users' activities.

Administrators

In order to manage the systems, support users, and monitor activities of the systems, QueensGate Grid administrators are logging in to Bearicade as an administrative account to be able to activate and deactivate user accounts, continuously monitor request and activity logs, and diagnose problems and giving support to the users of the system.

Bearicade administrators have multiple functionalities and features from within the dashboard as shown in table 4.1.

Table 4.1 Available Functionalities and Features to Administrator from Bearicade

Category	**Functionality**
User Information	Geographical Location
	Active Status
	Non-Sensitive Information
	2FA Token
User Management	Activate/Deactivate Account
	Delete Account
	Groups Assignment
	Developer Assignment
User Monitoring	Device Records
	Login History
	Activities
System	SSH
	File Manager
	Request Logs
	Settings
	Health Check

Developers & Researchers

Bearicade gives researchers the ability to access the data generated by Bearicade from the API, allowing them to investigate the data systematically research in order to establish novel findings in multiple fields, especially security. Developers also benefit from this functionality by developing various software and plugins such as visualisation application and machine learning models, which could be implemented in the core of Bearicade or as a third-party application for the users.

4.2 Data Collection

In order to collect data, this system was ready to have most of the core usage functionality and was set up to be used in the University of Huddersfield for final year undergraduate students and master students for the two modules: Parallel Computer Architecture Clusters and Grids, and Parallel Computer Architectures Cluster and Cloud Computing.

The students from both modules had to use the system as part of their module assignment; in order for the students to have access to the system, these are the steps followed:

1. Register for an account via the system registration page.
2. Once approved by system administrators, students would log in to the system.
3. Create job files.
4. submits HPC jobs to Cygnus and Condor.

The students can submit different types of jobs. However they are provided with templates for some parallel programs/jobs such as: (i) hello-world (ii) calculating PI (iii) matrix-matrix product (iv) matrix-vector product. Code 4.1, shows example of an uncompiled hello-world MPI C program. Code 4.2, shows the PBS job file used to run a compiled C code.

```
#include <stdio.h>
#include "mpi. h
int main(int argc, char ** argv) {
  int rank, size;
  MPI_Init( & argc, & argv);
  MPI_Comm_size(MPI_COMM_WORLD, & size);
  MPI_Comm_rank(MPI_COMM_WORLD, & rank);
  printf("Hello world from process %d of %d\n", rank, size);
  MPI_Finalize();
  return 0;
}
```

Code 4.1 Example of Hello-World MPI C program

```
#!/bin/bash
#PBS -N mpi_job
#PBS -l walltime=01:00:00
#PBS -q queue_name
#PBS -l select=2:ncpus=36:mpiprocs=36

mpirun ./hello-world
```

Code 4.2 Example of PBS job file

Steps 2-4 were repeated multiple time by the students throughout the year from different devices, locations, and browsers. Students had experienced most functions within the system, and data was logged from every activity.

4.3 Flexibility: Cloud Deployment

Bearicade includes Ansible playbooks that automates the deployment of a complete Bearicade stack on a variable-sized cluster, as explained in the deployment section of the Bearicade chapter. The following configuration allows Bearicade to be easily deployed in the cloud, such as a virtual cluster or on a bare-metal cluster. This feature has come with many advantages, as will be described below.

Deployment on the cloud has helped developers, researchers and students to work on projects and assignments by running their own instance of the Bearicade stack on a cloud provider such as Google cloud platform (GCP), Amazon Web Services (AWS), and Microsoft Azure.

4.4 Bearicade Performance Evaluation in The Pandemic

During the COVID-19 pandemic in 2020, nation-wide University campuses were shut; this made it difficult to manage and maintain the traditional cluster and thus making the preparation of the cluster for teaching a prolonged process. Research projects have also been impacted by the pandemic. Adding to all that, during the same year, there were

attacks that targeted HPC centres around the United Kingdom. All of the factors above has brought a number of HPC clusters to a halt. Fortunately for the University of Huddersfield, running Bearicade has enabled the users of the HPC clusters to be able to continue the work remotely without an interruption through the web dashboard. Administrators were able to continue monitoring and managing the systems through the equipped dashboard explained in section Dashboard Features and in case of system malfunctions, Bearicade's containerized architecture allow administrators to redeploy and reconfigure the systems easily.

4.5 System Usability: SSH vs Bearicade

As explained in this chapter, some of the University of Huddersfield students have been using Bearicade as their primary access gateway to QGG HPC resources. The students have also used the traditional method (SSH) to access their hand-built system as part of their coursework. This teaching delivery method gives the possibility for assessing the usability of Bearicade compared to SSH to determine how simple the system is to use and satisfaction and user attitudes toward the system. Therefore an assessment methodology is needed. System Usability Scale (SUS) (Brooke, 1995) has been used in order to measure these aspects.

Table 4.2 System Usability Scale Score for SSH and Bearicade

	SSH	**Bearicade**
Number of Users	37	12
Average SUS Score	53.8	64.0

The SUS score is a value between 0 and 100, demonstrating the system's usability; the higher the value, the better the usability. The same users were given a survey containing ten questions about each of the systems. Table 4.2 shows the average SUS score for each of the surveys. Bearicade survey involved 12 users scoring an average SUS score of 64.0, and SSH users involved 37 users with an average SUS score of 53.8.

It is evident from the result that students preferred Bearicade over SSH. In addition, the Bearicade is a simpler access system for HPC compared to SSH and has a better user satisfaction.

4.6 Summary

This chapter has provided detailed information on how Bearicade was deployed at the University of Huddersfield QueensGate Grid. The deployment allowed the users to use Bearicade as the only gateway for HPC resources, in addition, administrators managed the systems and users from within Bearicade. This has allowed Bearicade to prove it's usability from the users' perspective and ease the collection of data from different sources within Bearicade to enable the expansion of the security systems via Artificial Intelligence.

The next chapter will explore different AI model implementations using Bearicade. In order for these implementations to work, AI models require data to be trained and evaluated with.

Chapter 5

Artificial intelligence-based tools for secure access to HPC resources

Bearicade is a novel data-driven framework designed, developed, and implemented as part of this research. All of the users' data that Bearicade collects are from web elements. The gateway to access the distributed systems is through Bearicade's web dashboard. In addition, every interaction with these systems is authorized and recorded by Bearicade.

A secure artificial intelligent system would be able to use Bearicade's capabilities in data collection and existing data inventory. Thus, make decisions at any point of the users' interactions with the systems to detect and deal with anomalies and enhance detection accuracy.

This chapter will explore the possibilities of implementing Artificial Intelligent techniques using the data gathered via Bearicade based on the system view and the user view. Each of the sections below will discuss different implementations of Machine Learning models developed over the past four years.

Section Initial Model Algorithm Design below will discuss some of the initial model designs implemented that has were not very effective; their refinement led to the successful model designs. Section Anomaly detection of user behaviour on Bearicade's login page and Classification of Malicious Linux Terminal Commands are based on a paper published (Al-Jody

et al., 2020) that uses the data generated from Bearicade to build two different models per system that could identify anomalies in users' behaviour. Finally, section Model Per User For System will explain a different model design that could predict the users' authenticity. Different model will have different evaluation model. However, models should be evaluated using an appropriate evaluation methodology using trained and untrained data, including data generated from a real-world examples.

5.1 Using Bearicade

Over 26 months with the current deployment of Bearicade on the QueensGate Grid with over 120 registered users, Bearicade has received over 4700 authorization attempts, 1400 of which were granted. As explained in Chapter 3, Bearicade collects information about the user from multiple sources; thus, the information collected over two years from 1400 sessions is large enough to build Artificial Intelligence solutions to detect anomalies. Table 5.1 shows aggregate information for QueensGate Grid users' Behaviour on Bearicade, including the Average, minimum and maximum for some of the parameters that will be used to identify users in this section.

Table 5.1 Aggregate Information for QueensGate Grid users' Behaviour on Bearicade

	Screen Size	**Geo-Location**	**Device**	**Terminal Interactions**	**File Manager Interactions**	**Account Lifetime**	**Total Logins**
Average	2	5	3	91	204	81	40
Minimum	1	1	1	2	1	3	2
Maximum	9	103	48	323	743	753	670

A typical work-flow of a user executing a command on the end systems via Bearicade would consist of 3 steps: (i) Authorizing the user (ii) Logging the activity of the user (iii) Sending the command to the end system to be executed. Figure 5.1 shows a possible AI integration for this work-flow above on Bearicade. The figure broadly explains the users' interaction with Bearicade in order for them to execute commands on the end system, such as an

HPC cluster. Grey elements represent the core components of Bearicade, while the green represents the suggested AI components to be implemented. Decision A would process the users' information and make a decision before authorizing the user to the system. Decision B could process the command sent to the system and make a decision before executing it on the end system. The figure does not represent the only possible integration of AI components. On the contrary ,these components could implemented at any point of the system.

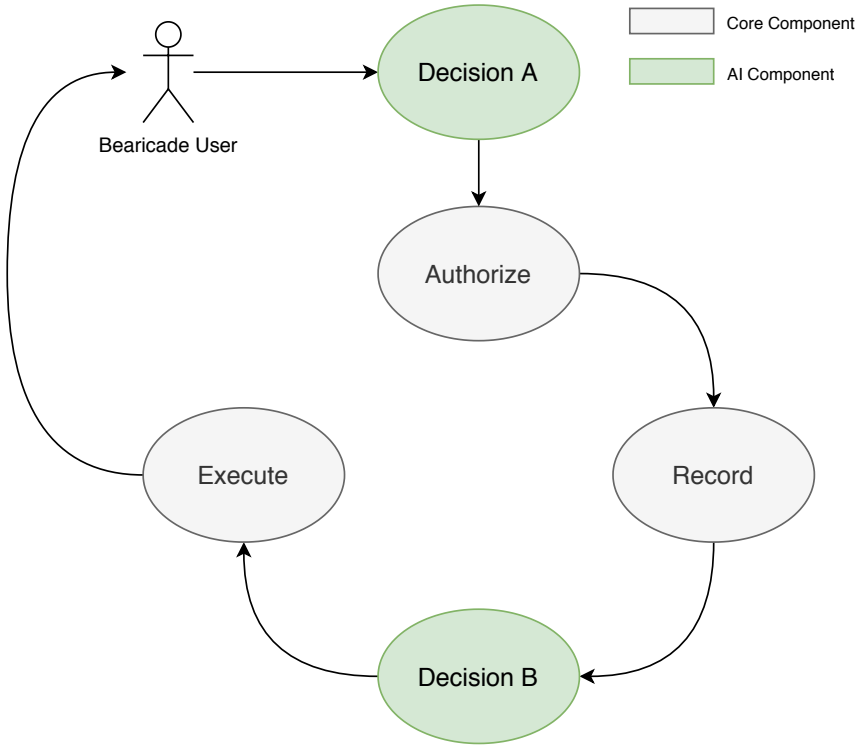

Fig. 5.1 Viable AI integration for Bearicade's components

5.1.1 Data Preparation

Although every Machine Learning models is different from the other, the approach for accessing and using the data will be remarkably similar. This section will focus on how the data collected and generated by Bearicade will be accessed, prepared, structured, and imported for use in the different Artificial Intelligent solutions.

All of the Artificial intelligence model implementations for Bearicade are developed with

the real data from The University of Huddersfield QueensGate Grid (QGG) main Bearicade instance. Some private instances of Bearicade were deployed to perform certain tests.

User Authentication

To have access to the data from Bearicade, developers can access such data from the pre-defined Representational State Transfer Application Programming Interface (RESTful API) actions. Developers could reach the RESTful API actions via the end-point address. In the case of QGG, its RESTful API address would be `https://qgg.hud.ac.uk/controller/api.php`. As discussed in section 3.2.6, with administrators' ability to assign API access keys to every developer wanting to access the RESTful API, developers were given access to specific actions required through the end-point for development.

Information Collection and Anonymization

As a developer request an API, an API key with zero permission will be issued to the developer. Administrators could then assign one or more actions permissions to the key as described in section 3.2.6.
Although the actions described in section 3.2.6 cover most of the database's information collections, administrators could also set up their own actions as permission to the developers. This is advantageous, as administrators could prevent users' sensitive information from being passed through the RESTful API. In other words, users' personal sensitive information will be anonymised, and only the required data will be passed.

Data Structuration and Importation

As explained in section 3.2.6, Bearicade uses JavaScript Object Notation (JSON) as its default transporting format for the RESTful API output data. This allows cross-language compatibility in gathering the data from Bearicade's RESTful API. Below are snippets codes demonstrating the ease in gathering the data from Bearicade's RESTful API.

```
curl --request GET \
--url 'https://qgg.hud.ac.uk/controller/api.php?a=listUsers' \
--header 'api_key: my_key' --header 'content-type: application/json'
```

Code 5.1 Sample API Request in Bash

```
import requests
url = "https://qgg.hud.ac.uk/controller/api.php"
querystring = {"a":"listUsers"}
payload = ""
headers = {
    'api_key': "my_key",
    'content-type': "application/json"
    }
response = requests.request("GET", url, data=payload, headers=headers,
 params=querystring)
```

Code 5.2 Sample API Request in Python

```
fetch("https://qgg.hud.ac.uk/controller/api.php?a=listUsers", {
  "method": "GET",
  "headers": {
    "api_key": "my_key",
    "content-type": "application/json"
  },
  "body": false
})
```

Code 5.3 Sample API Request in Javascript

```
CURL *hnd = curl_easy_init();
curl_easy_setopt(hnd, CURLOPT_CUSTOMREQUEST, "GET");
curl_easy_setopt(hnd, CURLOPT_URL, "https://qgg.hud.ac.uk/controller/api.
 php?a=listUsers");
struct curl_slist *headers = NULL;
headers = curl_slist_append(headers, "api_key: my_key");
headers = curl_slist_append(headers, "content-type: application/json");
curl_easy_setopt(hnd, CURLOPT_HTTPHEADER, headers);
```

```
8  CURLcode ret = curl_easy_perform(hnd);
```

Code 5.4 Sample API Request in C

5.1.2 Models Perquisites

The next chapter will explore Artificial Intelligence implementations using Bearicade's data. These implementations will require perquisites such as Artificial Intelligence libraries. Two main Artificial Intelligence libraries were used in these experiments. Undoubtedly, each implementation will have more unique perquisites; however, that will depend on the experiment.

Brain.js

Brain.js is an open-source Neural Network Library written in Javascript. Brain.js is highly customisable, meaning Neural Network could be tweaked to match desired accuracy (BrainJS, 2016). A great benefit of using brain.js is that all prediction could be calculated and run from the browser once the model has been trained.
Brain.js could also run server-sided through node js (Teixeira, 2012). Brain.js provides different types of Neural Networks, such as Feedforward Neural Network with backpropagation (w/o GPU), Time Step Recurrent Neural Network (RNN), Time Step Long Short Term Memory Neural Network (LSTM), and Time Step Gated Recurrent Unit (GRU).

TensorFlow.js

TensorFlow.js is another open-source Web Graphics Library (WebGL) accelerated Javascript library for building Machine Learning models. Being part of the TensorFlow ecosystem, models built are cross-compatible between languages such as Javascript and Python.
Like Brain.js, TensorFlow.js allows models to run through any Javascript runtime engine (Browser, NodeJS, Deno).

5.2 Initial Model Algorithm Design

Over the past four years and throughout the development of Bearicade and its Machine learning model, there have been many attempts to build a generation of models that could help identify anomalies in the system. Most of these attempts were unsuccessful or required improvements. However, the lessons learned from these designs led to building the current working model algorithms.
This section will briefly cover some of these attempts, going over the methods used to build the models and the results of these models.

5.2.1 Model Per System For Users

The Idea of a Model per System for Users algorithm is to have a model that could be trained with all users' data and able to identify which user is accessing the system depending on the login data passed. If a user not within the organization tries to access Bearicade, the model should be able to identify such an anomaly.

Data

Over four months, from the first release of the system, on January 19 with 50 student users of the system, a considerably large amount of data has been generated.

During the system's operation, the server has received 2030 login attempts, 536 of which are granted.

User's Login data are formatted as such:

Table 5.2 Users Login Example Row

User ID	Timestamp	fails	IP	Fingerprint	Reason
1	2017-11-07 07:00:26	1	0.0.0.0	859de64fe9...	Invalid Authentication Code

As previously declared in the Data Acquisition section, more data are gathered about the user from different sources. This data would be used later to improve the AI-based tools in Bearicade.

Another vital table to be used in the AI tools is the Users Activity. There has been a total of 16672 activity executed on the system, with an average of 340 activity per user. An activity would look like the table below.

Table 5.3 Users Activity Example Row

Activity	User	Type	Attempt	Flag	Success
1	1	Terminal	Command:"cd /home/root/; ls"	Path Manipulation	false

Data Preparation

Many of the Machine Learning algorithms do not accept categorical data as it is an efficiency limitation rather than the difficulty of implementation within the algorithms themselves. Categorical data are data with labels such as the key: Operating system, and its values: Windows, or Mac. These values are usually called nominal data.

Nominal Data should be converted into numerical data before feeding it to the ML algorithm. There are various ways to form the data gathered to the Machine learning algorithm to train it. Two of the most popular ways to convert the data are:

1. Label Encoding.
2. One-Hot Encoding.

Label encoding, also known as Integer Encoding, is a process of converting values into numerical values. For example, Windows becomes 1, Mac becomes 2, and Linux becomes 3.

This encoding method has a natural ordered relationship between its values, and ML might be trained to understand this relationship.

One-Hot encoding is used when there is no natural order relationship between values, and different values might be shared with different labels. An example of using this type of encoding would be when we have different users using multiple operating systems, then a matrix-like table could be generated:

Table 5.4 One-hot encoding

User ID / OS	Windows	Mac	Linux	Encoded Output
1	0	1	1	0, 1, 1
2	1	0	0	1, 0, 0
3	0	0	1	0, 0, 1

Model

The current implementation of ML has only been using the following data keys:

1. User ID.
2. IP.
3. Fingerprint.
4. Login Timestamp.

Upon initialisation of the model, the system request up to data from the database. A JSON of the data is formed, grouping it by the users' ID.

```
{
1: {
    "ip":["1.1.1.1","1.1.1.0", ...],
    "fingerprint": "bdd8d1aea7db15f2ec","4d35856e31c95dc9cc4", ... ],
    "timestamp":["2019-03-27 15:41:56","2019-03-28 14:41:01", ... ]
    },
2: {...}
}
```

Code 5.5 raw training data

To develop the one-hot encoding method, index dictionaries for the IP, fingerprint, and timestamp were created. These dictionaries are just arrays with a unique value in every index.

Brain JS requires a special format for the data to be trained; this format should be an object with two more objects inside (input, being the inputted data, and output, being the expected output). The system, therefore, generates a formed training data that is still unserialized, meaning that it requires encoding. This data would look as such:

```
{
    1: {
        input: {
            ip: ["1.1.1.1", "1.0.0.1", ...],
            fingerprint: ...
        },
        output: [1]
    }
}
```

Code 5.6 unserialized training data

As it is not acceptable to have an object in the input field, therefore as a final step, the system uses the dictionaries made formerly to encode the data and separate IP, fingerprint, and timestamp from each other for every user. The encoded concatenated data in the input array and the output array consists of the user id inside an array as shown in code 5.7.

```
{
    1: {
        input: [1, 0, 0, 1, 1, 1, 0, ...],
        output: [1]
    }
}
```

Code 5.7 serialized training data for user 1 to predict only IP

Three identical neural networks have been set up in figure 5.2 for the three values (IP, fingerprint, time). The neural networks were all configured using the following parameters:

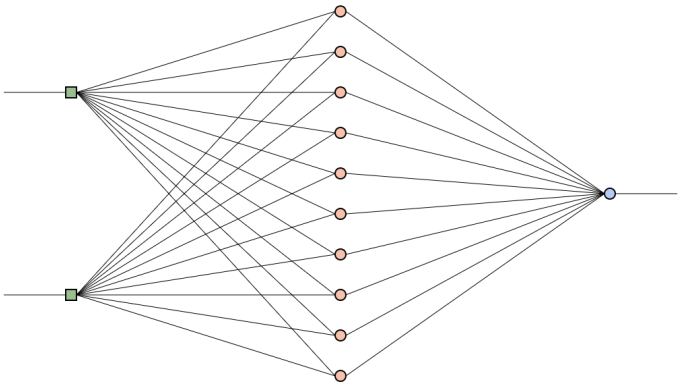

Fig. 5.2 Neural Network

Learning rate controls the speed of training the higher, the faster (0 to 1). Faster training might cause constraints on training results, causing new training to overfit. Similar to the learning rate, momentum is only multiplied by the next level's change value.

The network training might stop if the training error has gone below the threshold or the maximum number of iteration has been reached.

Training

After formatting the prepared data, the data was fed to the network to be used in training. However, feeding 150 data sets (3 variables x 50 students) would not be enough to increase the accuracy of the network.

The network had to be trained with random data from the training data set. Therefore, a function was created to generate genuine login data (IP, fingerprint, and time).

The network was then trained with one million genuine data records; after every 100 record was training, the accuracy of the model was calculated.

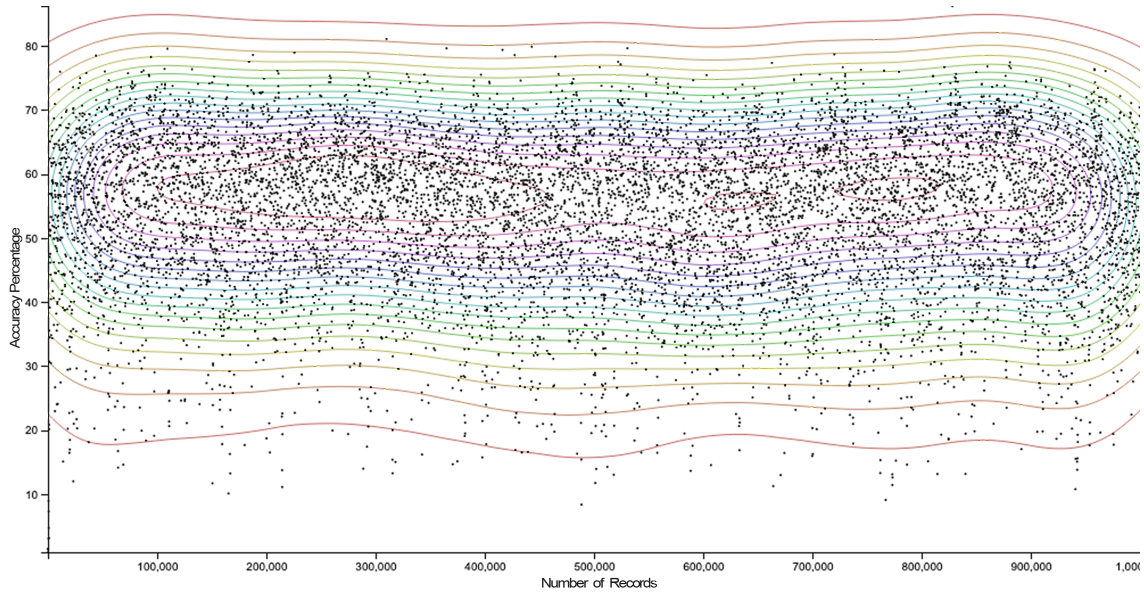

Fig. 5.3 Training ML with a Million Records

Results

Representing the accuracy as percentage in the y-axis and training records number in the x-axis, figure 5.3 shows the result from the training. The result has shown that the average accuracy of the network over 1 million records trained was 53.8%, with a peak accuracy of 86.2%.

Conclusion

The network was only trained with three variable with no relationship between the variable. Also, the records did not include disingenuous data. Moreover, Neural Networks configurations were not altered. These factors play a significant role in lowering the accuracy as the system cannot identify rogue records. Based on the outcomes of this approach, it was decided to develop a "Model per User" AI-based tool.

5.2.2 Precursory Model Per User Algorithm

There have been different approaches developed to get to the successful model discussed in section Model Per User For System. These approaches were not efficient nor successful in building an accurate model. Below are some of the algorithms that were attempted:

1. One-hot encoding: the successor model has used label encoding to encode the raw data before training the model by converting each value to a unique integer. One-hot encoding has been used; however, the training error was high due to the noise caused by the large binary values.

2. Model Per Information Per User: the successor model has encoded all the information shown in table 5.7 into a single object before training the model. Model Per Information Per User Algorithm has used an independent model per information. The idea was to multiply the results from each model by an independent weight then sum the results in order to get an accurate value. Although such implementation has provided the ability to control the importance of an information factor over the other, there was not much variety in the data in order to form a reliable prediction.

5.3 Classification of Malicious Linux Terminal Commands

5.3.1 Introduction

In any distributed systems, terminal languages such as bash are usually the link between the users and the local systems. Once a malicious user has managed to get access to Bearicade's dashboard or any shell access gateway by impersonating a user, the attacker would be able to communicate to the local systems and may cause harm to the local systems or other users.

This experiment aims to build an effective artificial intelligent method for analysing users' commands sent to the local systems and detect malicious commands. As shown in figure

5.1, this implementation would be part of decision B. The commands sent are analysed and given a score of suspicion before getting executed in the local systems.

5.3.2 Data

As explained in section 3.2.5, activities of the users' interaction with the local system are recorded before getting sent to the server. The information collected shown in table 3.3 includes the following:

1. User ID: Identification number for the user executing a command.
2. Date: Date and time of the activity recorded.
3. Component: Which component the user is interacting with, i.e. (File Manager, Terminal, System).
4. Local System: The local system the user is interacting with.
5. Command: The command or activity the user has attempted.

In a typical HPC environment, user commands are often limited to (i) interaction with the user's file system located at the user's home directory, e.g. (editing, creating, or removing files) (ii) interacting with the user's Jobs, e.g. (submitting, removing, querying status). (iii) interacting with certain applications. The following limitation makes it possible to detect and block commands that do not fit within these HPC environment limitations.

Data Preperation

At the time of doing the following experiment, Bearicade has recorded over 30,000 activities from the users. The data were filtered to exclude commands that are out of the HPC environment scope discussed above.
In order to create the dataset used for training, the filtered data were used to generate a

dataset containing the commands ran and their occurrences (a_n).

Where

$$a_n = \left(\frac{Command\ Occurrence}{Total\ Commands\ Count} \right) \tag{5.1}$$

Using the calculation above, the occurrences of a complete command (c) sent to the server would be calculated word by word. The probability of suspicion ($P(c)$) for the complete command consisting of a number of words (n) would be calculated using the following equation:

$$P(c) = \left(\frac{a_0 + a_1 + a_2 + ... + a_n}{n} \right) \tag{5.2}$$

5.3.3 Training & Evaluation

Using TensorFlow, a Convolutional Neural Network model was configured with an input layer size that would match the training dataset's input dimensions and an output layer size of 2 to match the Neutral and Suspicious predictions. Several hyperparameters varieties were tested to reach an optimal accuracy with a large number of different commands.

Subsequent to training the model, test commands were produced to test the accuracy of the model. As shown in table 5.5, the commands tested included typical ordinary HPC user actions such as interactions with personal files or job submission (C, D, E, F); other commands included modifying file systems (A, B).

Table 5.5 Classification results of Linux commands (Al-Jody et al., 2020)

	Command	Expected	Neutral	Suspicious
A	sudo rm -rf /	Suspicious	6.76%	93.24%
B	apt install python3	Suspicious	1.68%	98.32%
C	qsub helloworld.job	Neutral	99.99%	0.01%
D	cd ..	Neutral	97.56%	2.44%
E	rm ./hello	Neutral	96.52%	3.48%
F	rm -rf ./hello	Neutral	76.95%	23.05%

The results from the table 5.5 demonstrate the model's success from the high accuracy of this model. The model successfully identified different commands with different expected outcomes. This model is efficient for use in a real-world scenario where the users' commands are tested before being passed to the designated server. This approach, however, does not analyse commands that run from a script file or any executable example. To mitigate this issue, a number of different approaches could be done, such as (i) Enforce a policy to disable script execution (ii) Examine script files step by step before executing them.. At the University of Huddersfield QGG, script file executions are disabled by default. However, analysing script files would be considered a more viable approach.

5.4 Anomaly detection of user behaviour on Bearicade's login page

5.4.1 Introduction

The number of impersonation attacks on authentication systems has been increasing. The authentication layer of any system is the most crucial layer of all the system. Once an impersonation attack has been executed, the impersonated individual has lost all their privacy on the targeted system at that moment. In order to prevent such impersonation attacks, a profile regarding the legitimate user's behaviour should be collected and analysed and collected.

The idea of this experiment is to collect users' behaviour when accessing the system. By training a Machine Learning model to detect such behaviour, the model would be able to detect automated types of impersonation.

5.4.2 Data

Bearicade database contains a large collection of information regarding each of its user's behavioural data as explained in section 3.2.5. The data collection that represents unique behavioural characteristics for each user will be the following information:

1. Browser User Information: Details about the browser, such as the user agent.
2. Time: time spent on the page.
3. Mouse Clicks: click count along with the click details such as the timestamp of the click and X and Y coordinates for each click.
4. Mouse Movements: Includes the timestamp and the X and Y of the movement destination.

In order to analyse the data, some visualisation had to be setup. A heatmap of Bearicade's login page with the clicks and movements of a user was the best option to visualise such information. As shown in figure 5.4, the heatmap represent a random legitimate user accessing QueensGate Grid system via Bearicade's login page.

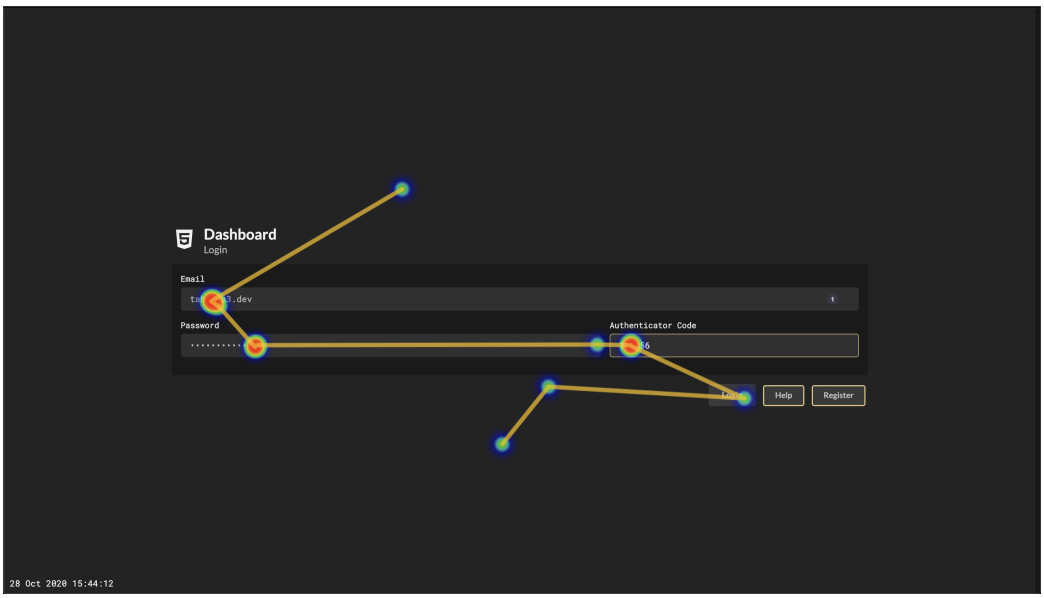

Fig. 5.4 Mouse events heatmap of a random user accessing Bearicade's login page

A number of random user's heatmap were analysed visually and found that the majority of users mouse movement were very similar, noting that the minimum, maximum, and average were within close proximity. This could reduce the accuracy of the Neural Network's model if no more features were added to the model.

5.4.3 Training & Evaluation

The dataset that was pulled from Bearicade was treated as a neutral dataset for training. Emulated suspicious data were generated to match logins from bots and additional irregularities. The size of both sources matched to keep a fair bias in training.
Feed-forward Neural Network using Brain.js were configured with an input layer to match the input size, along with an output layer that matches the normal and suspicious predictions. A number of hyperparameters were tested until an optimal output was reached.

Table 5.6 Classification Results of User Behaviour (Al-Jody et al., 2020)

Test Case	Expected Output	Normal %	suspicious %
A	Neutral	99.72%	0.28%
B	Suspicious	2.74%	97.26%
C	Neutral	91.34%	8.66%
D	Neutral	99.79%	0.21%
E	Suspicious	1.81%	98.19%

The trained model was tested with different test cases from both outsides and within the training dataset. As shown in table 5.6, test case (A and B) were from within the training dataset, while test case (C) was through a generated random input, (D) through Neutral human input, (E) programmable suspicious input. The accuracy of the model was high for all 5 test cases.

5.5 Model Per User For System

5.5.1 Introduction

There are many features that could be analysed from the user's behaviour that would allow a system to estimate the probability of the user's authenticity. The purpose of this work is to build a Neural Network model for each user of the system to calculate the probability of the users' authenticity from different parameters passed. Such implementation would fall in Decision A as shown in figure 5.1.

5.5.2 Data

As discussed in the section above, the different parameters that would be passed to the model would include the following information shown in table 5.7.

Table 5.7 Users Login Example Row

Information	**Examples**	**Details**
Timestamp	2021-02-02 15:14:11	Time and date of attempt
Geographical Location	United Kingdom	Geographical Location from IP Address
Operating System	Mac OS	Operating System of current Attempt
Platform	MacIntel	Device Platform used in Attempt
Screen Size	1920x1080	Screen Resolution of the Device
Browser	Chrome	Browser used in attempt

Each set of this information represents unique features that could identify a particular user on a Bearicade. Each of the users on QueensGate Grid Bearicade logs into Bearicade using different parameters shown in 5.7, graph 5.5 illustrate the data distribution between Bearicade users showing device used (Version), Geographical Location (via IP), and screen size. The graph shows a more consistent parameter distribution when compared to the other parameters.

Table 5.1 shows that on average, a Bearicade user authenticates to Bearicade 40 times, and the account is used on average over 81 days per user. Also, users use two unique screen sizes, five different IP addresses, and three different devices used to login to Bearicade.

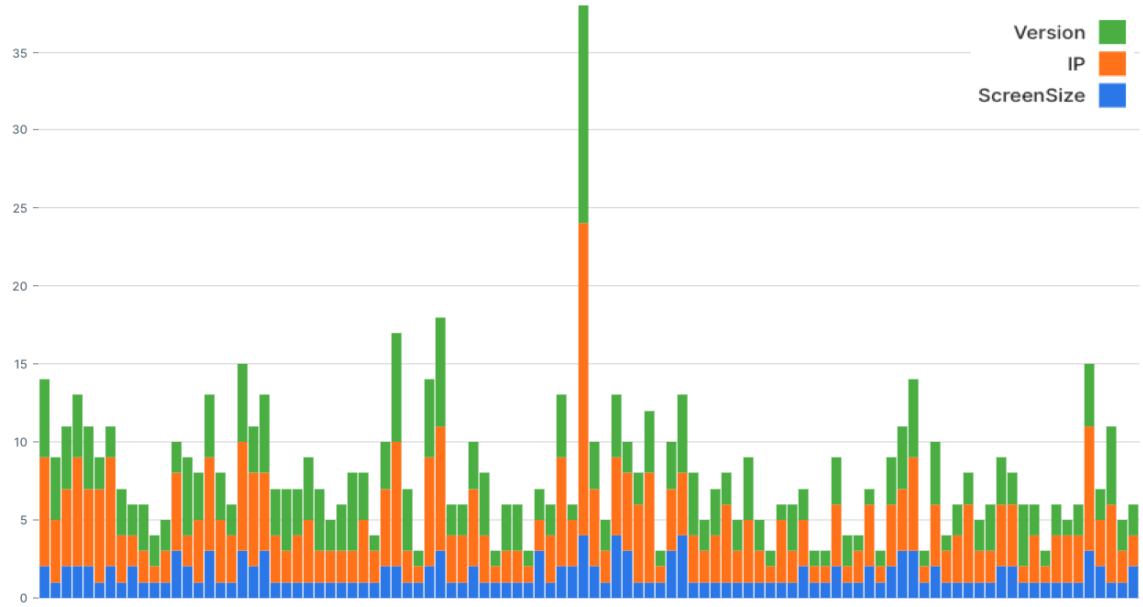

Fig. 5.5 Unique data distribution between users

Data Preparation

As the data gets requested from the Application Programmable Interface (API), a JSON array of objects gets parsed, forming the raw data for a particular user, which is required to train the NN model.

A sample of one of the objects in the array is shown in 5.8. However, this model will only use four keys in the object, which will be (i) timestamp (ii) browser version (User Agent) (iii) browser IP (iv) screen size.


```
{
"id": "1",
"timestamp": "2021-02-02 04:14:11",
"browserCookies": "authID=e8f7dc63331e43151c1d379ba343052737728b01",
"browserLanguage": "en-US",
"browserName": "Netscape",
"browserPlatform": "MacIntel",
"browserVersion": "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_1_0)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36
",
"browserIP": "92.xx.xx.xx",
```

```
    "historyLength": "3",
    "javaEnabled": "0",
    "referrer": "https://qgg.hud.ac.uk/access/",
    "screenSize": "1920x1080",
    "fingerprint": "c919bb27503f12dbd6528891c464383c",
    "loginCount": "1"
}
```

Code 5.8 Raw User Browser Data

Such data shown in 5.8 will require to be encoded before training the Neural network. The process of information encoding will be explained in the following sections.

Time-Stamp (Date and Time)

The timestamp is the date and time a user has logged in, and the information has been recorded. Time and the day of login could form unique identifiable information to help recognise user's habit of using the system throughout the days and time of the week. Since date and time is a piece of historical information, only the time (Hour) and the day of the week will be extracted from the timestamp.

The data will then be used to form a matrix with 24 columns to represent the time of the day and seven rows to represent the day of the week. An encoding function would be passed the timestamp and return the location of the time and date in the matrix. Figure 5.6, shows a heat-map visualized matrix for a random user. By visually analysing the heat-map, multiple features could be extracted. For example, the user does not have the same frequency of login during Saturday and Sunday compared to the rest of the weekdays. Also, the user has a certain time range during weekday where the user logs in.

Geographical Location

Over 95% of QueensGate Grid users have used Bearicade from within Great Britain only. However, some users tend to access Bearicade from outside Great Britain. Thus, the country

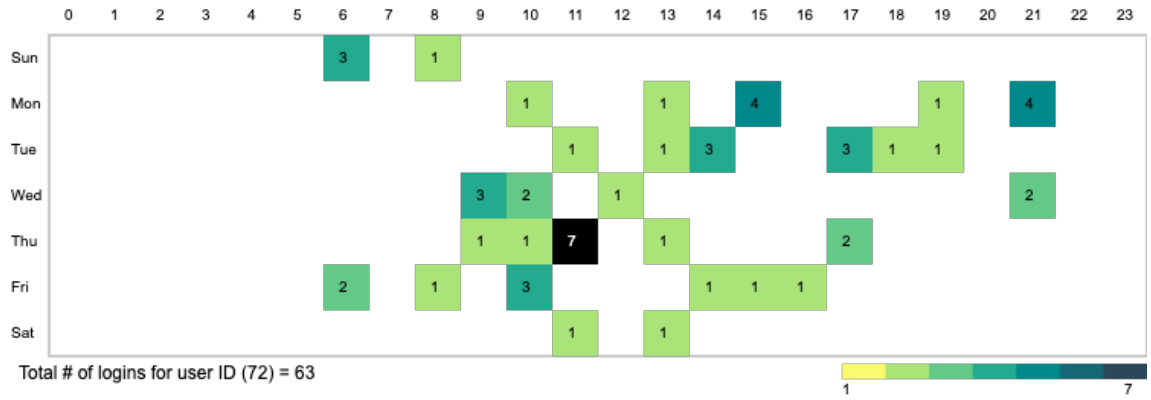

Fig. 5.6 Timestamp Matrix Heat-map visualization

a user trying to authenticate from should be considered in determining the user's legitimacy. The only way Bearicade could determine the geographical location of the location is through the IP address of the user. Bearicade uses an IP lookup database that is used to determine the country of origin of the IP address. As shown in 5.9 Bearicade also uses an object containing a list of all 240 countries with IP address allocations sorted by distance from Great Britain. The list would then be used to translate the country into a number.

```
{
"gb": 1,
"ie": 2,
"be": 3,
"nl": 4,
...
"nc": 238,
"nf": 239,
"nz": 240
}
```

Code 5.9 Countries List with IP addresses allocations sorted by distance from GB

Screen Size, Operating System, Platform, and Browser

When a user logs in to Bearicade, the Screen Size, Operating System, Platform, and Browser are often linked together, representing the devices a user is logging in from. This could have a substantial positive influence on the accuracy of the model.
The Screen size and platform are gathered directly from the user's browser along with the user agent, which might look something like 5.10, the user agent is parsed to give details about the operating system and the browser the user is using.

```
1    Mozilla/5.0 (Macintosh; Intel Mac OS X 11_1_0) AppleWebKit/537.36 (KHTML
    , like Gecko) Chrome/87.0.4280.67 Safari/537.36
```

Code 5.10 User Agent Example

All of this information is encoded similarly. Using a label encoding method, a dictionary is created with the possible information varieties turning the information into a unique number representing the information. In case a new variety is not presented in the dictionary, it gets added to the dictionary then encoded through the same method.

Anomaly Data

All the methods above are used to prepare genuine data for model training; the model requires anomaly data in order to be able to differentiate between what is considered to be safe or not. Since the system does not have enough unsafe data, a method of generating anomaly data is required for the training process.

Generating such data is done using the dictionaries generated as explained in Data Preparation. However, anomaly data should not consist of elements a user has used before, which would be considered safe. Thus while preparing the genuine data, an array of safe elements would be recorded; elements in this array would then be eliminated from the main dictionary to create a new dictionary for generating anomaly data.

5.5.3 Training the Model

As the dataset has been compiled from the steps explained above, the next process is to train the model with datasets generated.

Model Validation

A model validation techniques need to be used in order to evaluate the effectiveness of the model, especially on new data that the model has not been trained with before.

Fig. 5.7 5-fold cross validation sets

Cross-Validation is a statistical method that was used to test the accuracy of this model. Cross-Validation takes a k-fold parameter which is the number of groups that the dataset will be split into. Figure 5.7 shows 5-fold cross-validation, which duplicates the dataset into five different sets; each will split the data into five folds. 20% of the total dataset one of the

fold will represent the testing data, and the rest of the 80% will be the training data from different sections of the original dataset.

After forming the k-fold sets, a number (k) of Neural network models will be created, each being trained then tested with the different sets generated from the folds.

To summarize the training process, the original dataset processing will go through the following steps:

1. Generate the dataset from Bearicade data and the anomaly data.
2. Randomly shuffles the dataset.
3. Generate k number of sets each with k number of folds.
4. Hold one of the k folds as a testing data.
5. Train the remaining folds from each set into a different model.
6. Evaluate the testing fold on the trained model.
7. Measure the k models performance from the training and testing data.

Model Configuration

Artificial Neural Network has a number of parameters that must be set when configuring the neural network for training. Such parameters are called hyperparameters. These parameters affect the network's architecture and thus have an impact on the effectiveness and performance of the Neural Network. Table 5.8 shows the different hyperparameters used in the configuration of the final most optimal neural network build for this model.
As shown in table 5.7, the model has six main inputs, and thus the input size was set to 6. As for the output layer, the prediction should include a percentage of safe and a percentage of unsafe and thus, the output size was set to 2. Multiple configuration for the hidden layer were tested and found that a 3 hidden layer first layer with 6 nodes to match the

Table 5.8 Model Configuration Parameters and Values

Parameter	Value
Input Size	6
Output Size	2
Hidden Layers	3
Nodes per Hidden Layer	6, 4, 2
Activation	Sigmoid
Binary Threshold	0.5
Error Threshold	0.005
Learning Rate	0.01
Cross Validation Folds	5

input layer, second was 4 nodes, and the third layer was set to 2 nodes to match the output layer. Such configuration visualized in figure 5.8 were found to be the most optimal. The

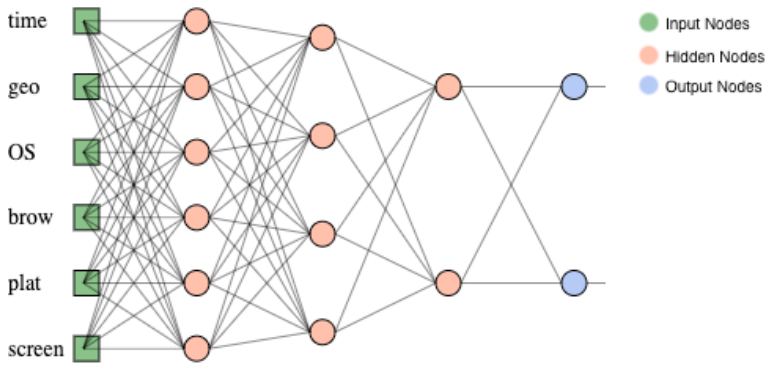

Fig. 5.8 Neural Network model layers visualized

output values should be a value between 0 and 1 to represent a percentage; thus, one of the most suitable activation functions to choose from would be the sigmoid/logistic function or softmax function. Due to the limitation of Brain.js, the library used to build this model, and only sigmoid could be used to build this network. The Binary Threshold is a value set in order to help with the testing of the model. A 0.5 value indicate the midpoint between safe and unsafe data. For this example, 0.5 indicated that any value larger than 50% would be considered a 1 (100%) and any value lower or equal would be considered a 0 (0%).

As for the Error Threshold, which indicated the optimal low value of the error, iteration would be stopped if reached. Although a 0.005 error was not reached in this model, a 0.006 value was reached; it allows the model to be further trained and may lower the error percentage.

The learning rate indicates the frequency of the weights getting updated while training. The learning rate had the most significant impact on this model's accuracy; different values for the learning rate have tested these values were between 0.5 and 0.0001. The test was done by building the neural network model using the same dataset and the same order using these different learning rate values (0.5, 0.1, 0.01, 0.005, 0.001, 0.0005, 0.0001). Graph 5.9, shows the different learning rate values plotted to show the error values and the same number of iteration.

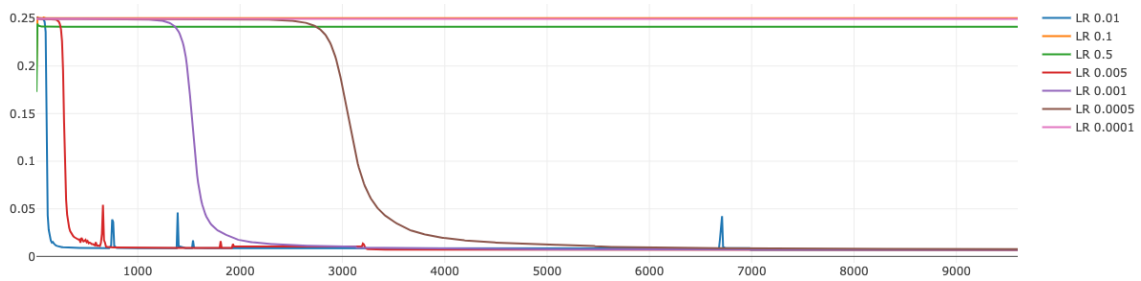

Fig. 5.9 Different Learning Rate tested with error

As shown in table 5.9, the different learning rate along with the error values was ranked from the most efficient and accurate to the least. The test showed that a learning rate value between 0.01 and 0.0005 was the most optimal. Since the 0.01 learning rate value scored the best regarding the error value, it was chosen for this model as an optimal value.

Finally, as for the cross-validation folds explained in Model Validation, five-folds were used the most as it gives a good split between the training data and the testing data. However, some users with lower datasets required higher folds as explained in Model Validation.

Table 5.9 Different Learning Rate tested ranks

Learning Rate	Error	Rank
0.5	0.2412088	5th
0.1	0.2502621	7th
0.01	0.006876066	1st
0.005	0.006898225	2nd
0.001	0.007095933	3rd
0.0005	0.007620678	4th
0.0001	0.2493396	6th

5.5.4 Evaluation

For every user, a unique model has to be created hence the model per user name. To evaluate this neural network model, multiple models need to be trained with datasets from different users.

Table 5.10 Users' data summary used to evaluate the model

User	Screens	IP addresses	Devices	Days of Usage	Total Logins
A	9	105	49	782	275
B	4	20	14	276	63
C	1	2	1	7	4

Table 5.10 shows the different users and count of their data that will be used in order to evaluate the neural network model. The data represent 5 different values

1. Screens and IP addresses: the count of different screen resolutions and IP addresses used to access Bearicade, respectively.
2. Devices: count of unique Operating systems, Platforms, and Browsers used to access Bearicade.
3. Days of Usage: the number of days between first and last login.
4. Total Logins: the total number of successful authentication requests made to Bearicade.

The users chosen represent different use cases of the users. User A is the most active user on Bearicade, with over 782 days of usage and 275 logins. User C is one of the least active users; this user still has data that could be used to build a model. User B is a midpoint and represent the users that have a short usage period on Bearicade with only seven days of usage and four logins. The purpose of choosing these different users is to evaluate how the model performs on these different usages and different dataset sizes.
All of the users' models were trained using the same network hyperparameters explained in the section above with 9600 iterations. The size of the genuine dataset is equal to the number of total logins. Anomalies data were then appended to the dataset, the size of the

anomalies data made equal to the number of total logins. For example, user A has 275 total logins (genuine data); thus, another 275 anomalies data would be added to the data forming dataset with 550 elements.

User A Model

User A dataset contained 550 data to be trained in the model. However, since the cross-validation method was used with 5-folds, only 80% of the data (440) were trained in 5 different sets, and the other 20% (110) were used in testing the model. As User A has by far the largest number of logins, the size of the dataset was adequate in training the model and resulted in low error percentages for all five sets compared to the other users, as shown in figure 5.11. As shown in figure 5.10, the average training error across the 5 different sets was 0.009916 (0.99%). Some of the sets had misclassified some of the test data, but this is acceptable as the anomalies test data may be completely new for the model.

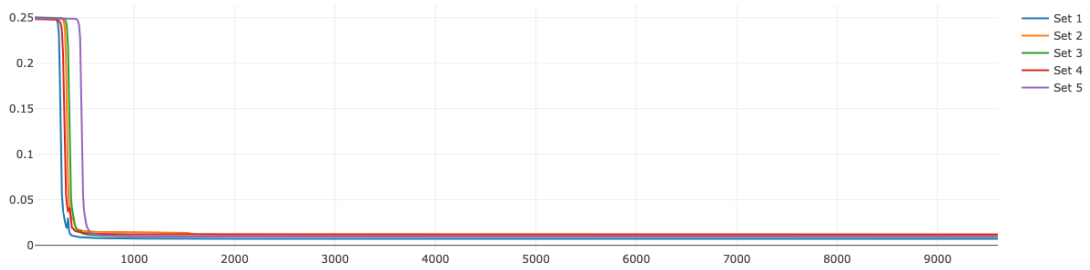

Fig. 5.10 Cross Validation for User A model

User B Model

Using the same number of the fold as the User A model, this model was cross-validated. User B dataset contained 126 elements split into 63 genuine and 63 anomalies. With only 23% the size of the dataset compared to the User A dataset, the model performed in an acceptable manner with 0.017806 (1.78%) training error with some misclassified test data in

Fig. 5.11 Evaluation for User A model

some sets. Figure 5.12 shows the cross-validation training error along with the iteration for the five sets, figure 5.13 shows the error percentages along with the misclassified test counts and train time for all five sets.

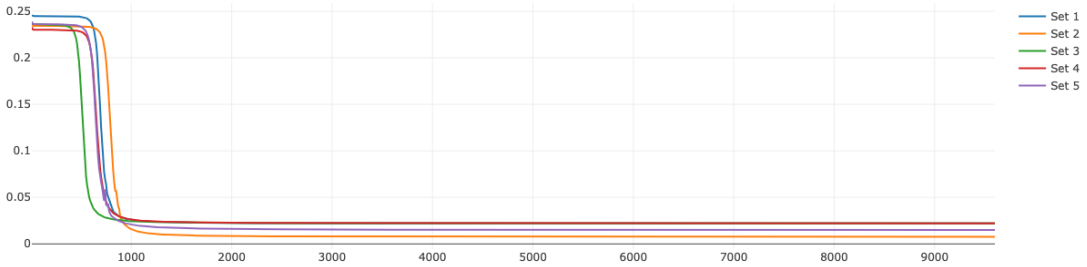

Fig. 5.12 Cross Validation for User B model

User C Model

Model for User C has the least training data with a size of 8. Thus, cross validation would not work on such a small data set. However, the model was still trained without folds to evaluate the training error for such a small dataset. As expected and shown in figure 5.14, the model had a high training error percentage of 0.222 (22.2%).

Fig. 5.13 Evaluation for User B model

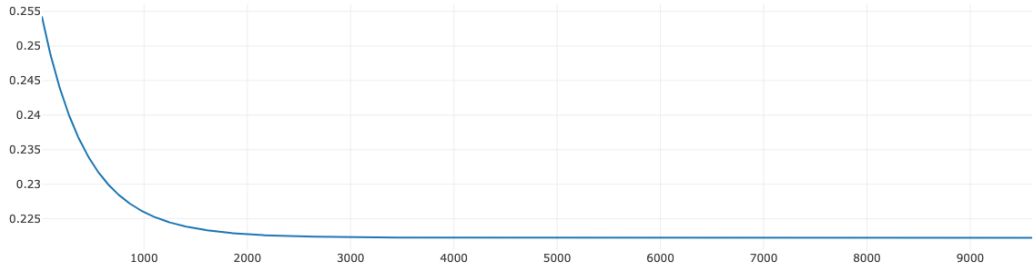

Fig. 5.14 Training error for User C model

5.5.5 Conclusion

To illustrate the accuracy of trained models, three different types of users were chosen, each with different dataset sizes. The model-per-user algorithm seems to give much better results compared to the initial algorithm designs. However, since the algorithm depends on the individual users' data, it requires a large number of data to have a valid model.

It is expected that a typical Bearicade user would be required to use the system for at least six months or have more than 60 successful logins to have enough data gathered about the user in order for the model to learn the user's behaviour and preferences.

To evaluate how these models perform in a real-world scenario, users A and B were asked to log in to Bearicade after their independent model was trained. In addition to these logins, a slight modification of the recent login data was modified to see how the model prediction

works compared to the original login. Moreover, anomaly data were generated to evaluate the model's prediction.

Table 5.11 Real-world test for User A and User B models

User	Test	Expected	Safe Prediction	Unsafe Prediction
A	Genuine Login	Safe	99.181%	0.817%
A	Genuine Login with modified Geo-Location	Unsafe	1.407%	98.592%
A	Genuine Login with modified Device	Unsafe	1.406%	98.593%
A	Anomaly Login Data	Unsafe	1.407%	98.592%
B	Genuine Login	Safe	99.086%	0.915%
B	Genuine Login with modified Geo-Location	Unsafe	2.956%	97.043%
B	Genuine Login with modified Device	Unsafe	3.124%	96.876%
B	Anomaly Login Data	Unsafe	2.956%	97.043%

Table 5.11 above shows how both User A and User B models have matched the new test data with the expected outcomes. With a high percentage of accuracy, the model could be an optimal solution to predict the authenticity of the user attempting to login to Bearicade, all from the data collected over time from the user.

5.6 Summary

This chapter presented a number of Machine Learning techniques developed to integrate into the Bearicade framework. The Machine Learning models and data used to train and validate the models applied for HPC security were presented. Three case studies were considered addressing the security from the point of users' behaviour such as the mouse movements and the interactivity with the portal and more comprehensive model that includes not only the user behaviour but also the environment variables based on the time, location and equipments used by the users.

The results obtained are encouraging demonstrating high accuracy in detecting anomalies and verifying users' integrity.

Chapter 6

Conclusion

For most information technologies and computing networks, cyber protection is paramount. High-performance computing is used in science studies, academia, and industry. HPC technologies are designed to take advantage of HPC systems' parallel architecture. Present research into High-Performance Computing systems is focused on improvements in application development, parallel algorithms, and computer system architecture. However, no significant attempts have been made to establish universal security requirements for High-Performance Computing. Security of High-Performance Computing infrastructure is often an afterthought of various administrative strategies that do not satisfy additional High-Performance Computing security requirements.

In the past, security breaches affecting HPC centres were relatively rare. However, recently the HPC system components' vulnerabilities are on the rise. Furthermore, the incentives for attacking HPC resources are increasing in value due to the substantial computational power, high bandwidth networks, and massive storage. For example, in the past, four years ago at the University of Huddersfield QueensGate Grid, there were, on average, 50,000 malicious login attempts that occur on a daily basis. As a result, increasing the protection of HPC systems is a necessity. According to a recent presentations by the experts from the UK Hight-Performance Computing Community, during the meeting of the HPC-SIG group,

there is no widely agreed security mechanism for the HPC systems of the HPC-SIG Group. The new security architectures for High-Performance Computing systems are proprietary, unreleased institutional solutions.

A comprehensive literature review of the existing access solutions for HPC systems revealed that although advancements are made towards HPC portal solutions, such solutions lack required basic security requirement implementations. Furthermore, the limited published security solutions for HPC have not been integrated into existing access solutions. Moreover, the current non-portal primitive access solutions use insufficient security practices in many HPC centres, causing attacks similar to the summer 2020 attack that targeted HPC centres in the United Kingdom.

The majority of cyber-attacks have a life span. Every stage of its growth can produce indicators that are difficult to detect by security professionals and system administrators using the conventional HPC solutions. As a result, it was essential to improve SIEM/SOAR systems. SIEM/SOAR systems provide a way to visualise and capture data and incidents in most system components, providing system administrators with a large volume of data to analyse. Because of the advancements in these systems, security administrators are inundated with data, much of which is noise, making attackers' mission easier. There is a need for SIEM/SOAR systems to be developed specifically for the HPC environment. Integrating artificial intelligent technologies in the SIEM/SOAR systems could vastly help system administrators in monitoring HPC systems activities. Data generated from HPC centres and their users is unique and differs from typical computer systems infrastructures. Therefore, the data generated from the HPC security system can be used to train artificial intelligence models for autonomous detection.

The aim of this research was to create a novel data-driven secure access framework for HPC systems that combines the capabilities of the Security Orchestration Automation and

Response System (SOAR) and Security Information and Event Management System (SIEM). The framework will allow further development of Artificial intelligent algorithms designed for anomaly detection in access and usage of HPC systems. To achieve the aim, (i) thorough research needed to be conducted to investigate the current challenges and solutions within HPC. (ii) it was necessary to design, develop and deploy the framework in the University of Huddersfield Campus Grid for management of HPC resources and users. (iii) analyse the data collected and accelerate the framework's security by building machine learning models for autonomous anomaly detection. (iv) evaluate the effectiveness of the framework.

As a result of the efforts outlined above, Bearicade was introduced. Bearicade is an HPC users and security management gateway; it is built based on an architecture that allows users to access several nodes on/off-premise remotely and includes a web terminal and file manager implementation. Bearicade is built on well-known browser features, and it allows for safe operation and access to HPC systems without requiring specialised user training. System security is accomplished by continuously tracking, analysing, and interpreting data, such as user interaction, server requests, access devices, and geographic locations, among other things. Anomalies in users' actions would be noticeable to server administrators, allowing them to take quick action to mitigate risks. In addition, the system does not use SSH as its primary remote access portal. Instead, the online dashboard serves as a portal. Bearicade, which is based on a RESTful API, helps administrators and developers visualise and analyse activities over the systems, allowing machine learning deployment to aid in enhancing systems security based on the data gathered.

Bearicade was deployed at the QueensGate Grid at the University of Huddersfield. Users used Bearicade as their only access to HPC services, and administrators monitored their systems and users from within Bearicade. This has helped Bearicade to demonstrate its versatility and make data collection from various sources easier inside Bearicade, allowing for the extension of security systems by Artificial Intelligence.

As part of this study, Bearicade is a data-driven architecture that was designed, developed, and implemented. Bearicade captures all of its users' data from dashboard components through the users' interactions. Bearicade's dashboard serves as a portal to the HPC systems. Furthermore, Bearicade authorises and records any interaction with these systems. An artificial intelligence system could be used with Bearicade in data analysis and internal data inventory. It can help to make decisions at any point during a user's interactions with a device to identify and respond to irregularities and improve detection accuracy.
Since the creation of Bearicade and the development of Machine learning algorithms, many efforts have been made over the last four years to create a generation of models that detect anomalies in the system. Many iterations of these models have contributed to the novel successful working models.

Various Machine Learning models were developed, each of the models serves as a use-case study. Three Machine Learning models were built. The first model was trained based on the perspective of user behaviour, such as mouse movements and dashboard interactivity. Second model identified malicious user commands before being sent to the target system. Finally, a detailed model used user behaviour and environmental variables dependent on time, place, and equipment used by users to train Artificial Neural Network models to detect anomalies in users' access to the HPC systems. Various tests were run on the QGG grid HPC tools to test the Bearicade framework's success in standard scenarios related to safe access to HPC systems. The findings are promising, showing high precision in identifying irregularities and checking the credibility of users, also opening up potential development for other Artificial Intelligent algorithms.

The novelty of the Bearicade framework is demonstrated in the framework's architecture and Machine Learning models developed through Bearicade. This ability that has not been previously available for conventional access methods in HPC and the uniqueness of the work discussed in this study. Bearicade is not just a prototype; it is a production system and effective solutions for authentication authorisation. This claim is proven by its four

years of active duty as the exclusive access software for the QueensGate Grid's users and administrators at the University of Huddersfield.

Addressing Research Questions

Following the extensive literature review conducted, there were some research questions identified. It is necessary to confirm the efficacy of the proposed framework (Bearicade) and the complementary Artificial Intelligent Machine Learning models developed. This section will address each of the research questions.

How can the technologies and data provided by this research contribute to the development and implementation of automated security access system in HPC?

The development and four-year deployment of Bearicade at QGG have allowed this system's usage as the primary access system for QGG users. The system's architecture has enabled the system to gather various information from its users forming a large dataset that was then used to build and implement multiple Machine Learning models serving as an automated anomaly detection over different components of the system.

To what extent such an implementation can protect HPC systems and authorised access?

Bearicade follows the OWASP Application Security Verification Standard and contains multiple security features that many other competitive systems lack; this is essential for a secure authentication and authorisation system access mechanism. In addition, the evaluation of the developed Machine Learning models has proven the models' capabilities in identifying anomalies in users' physical behaviour actions throughout the users' interaction with the system.

Will such implementation help in the Management and the usability of HPC systems?

Bearicade is currently used as the primary access gateway for QGG users, and the system has demonstrated that it can provide a smooth and automated operation in regards to the users' registration, authentication and user and job management. Also, the system has eliminated the need for users to have knowledge in Bash, that the SSH access method required. Finally, according to a conducted System Usability Score survey, Bearicade has a better usability and user satisfaction compared to traditional SSH.

Limitation

Bearicade has provided an efficient solution to a current problem and has contributed to the gap of knowledge. However, some limitations should be noted:

1. Bearicade's codebase is very large and requires continuous testing for any vulnerabilities that may be caused by one the component of the framework.

2. Since Bearicade's front-end is based on web technologies which is currently evolving rapidly, the system will require to be updated regularly to achieve cross-device compatibility and secure state.

3. Bearicade works well with Red-hat Enterprise based Linux operating systems. However, Bearicade was not tested on other operating systems such as Debian.

4. Although Bearicade has the basic functional features as a HPC system, it will require further development of add-ons to be able to replace the more advanced HPC-focused portals and improve the users' experience.

5. As with most Machine Learning models, some of Bearicade's models require a large dataset in order to achieve a reliable anomaly detection accuracy. In addition, Machine Learning models may introduce false positives/negatives.

Future Work

The work presented in this thesis, including Bearicade codebase and some of the Machine Learning models, has been published and is freely available to the community as open-source projects. Continuous monitoring of the system components will be carried and the project will be update accordingly to keep the system at a reliable and secure state. Bearicade has been implemented and tested extensively at tier 3 HPC systems (QGG). Additional efforts will be made to ensure the framework's compatibility with a wider range of HPC systems configurations and environments. This will allow the framework to be deployed at other HPC centres.

Further Bearicade add-ons will be developed to guarantee the framework's adequacy in replacing more advanced HPC-focused portals and better user's usability of the dashboard. Moreover, additional Artificial Intelligent tools and Machine Learning models will be explored to enable better anomaly detection accuracy and cover more system components to be tested for anomalies.

Future publications, currently in preparation, will further expand some of the work presented in this thesis. Our published journal article is exploring some of the work done in this thesis, and journal article is currently under review, with two more publications in progress extending the work presented in the PhD thesis.

References

Abramson, M. (2014a). Active authentication from web browsing behavior. In *Smart card alliance conference*, pages 1–16.

Abramson, M. (2014b). Learning temporal user profiles of web browsing behavior. In *6th ASE International Conference on Social Computing (SocialCom'14)*.

Abramson, M. and Aha, D. (2013). User authentication from web browsing behavior. In *The Twenty-Sixth International FLAIRS Conference*.

Abunadi, I. and Alenezi, M. (2016). An empirical investigation of security vulnerabilities within web applications. 22(4):537–551.

Activeeon (2014). ProActive from Activeeon.

Adaptive (2006). Products – Adaptive Computing.

Al-Garadi, M. A., Mohamed, A., Al-Ali, A., Du, X., and Guizani, M. (2018). A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security.

Al-Janabi, S. T. F. and Saeed, H. A. (2011). A neural network based anomaly intrusion detection system. In *2011 Developments in E-systems Engineering*, pages 221–226. IEEE.

Al-Jody, T. (2017). Bearicade. https://github.com/TA3/bearicade.

Al-Jody, T. (2018).

Al-Jody, T. (2019). web-user-behaviour.

Al-Jody, T. and Holmes, V. (2019). Security orchestration, automation and response (soar) in hpc. page 78. Emerging Technology Conference.

Al-Jody, T., Holmes, V., Antoniades, A., and Kazkouzeh, Y. (2020). Bearicade: secure access gateway to high performance computing systems. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1420–1427.

Alam, D., Zaman, M., Farah, T., Rahman, R., and Hosain, M. S. (2017). Study of the Dirty Copy on Write, a Linux Kernel memory allocation vulnerability. In *2017 International Conference on Consumer Electronics and Devices, ICCED 2017*, pages 40–45. Institute of Electrical and Electronics Engineers Inc.

Alpar, O. (2014). Keystroke recognition in user authentication using ANN based RGB histogram technique. *Engineering Applications of Artificial Intelligence*, 32:213–217.

Altair (2019). HPC Administrator's Control Center for Managing, Optimizing, and Forecasting Resources | Altair Control.

Altair Korea Enterprise Computing Division (2018). PBS Access & PBS Control.

Apache (2016). Apache Airavata.

Apostal, D., Foerster, K., Chatterjee, A., and Desell, T. (2012). Password recovery using MPI and CUDA. In *2012 19th International Conference on High Performance Computing, HiPC 2012*.

Apruzzese, G.; Colajanni, M.; Ferretti, L.; Guido, A.; Marchetti, M. (2018). *2018 10 th InternatIonal ConferenCe on Cyber ConflICt CyCon X: MaXIMISInG effeCtS CopyrIGht and reprInt perMISSIonS*.

Arbaugh, B. (2002). Security: Technical, social, and legal challenges. *Computer*, 35(2):109–111.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing.

Atos (2010). Bull Extreme Factory.

Bace, R. and Mell, P. (2001). Intrusion detection systems. Technical report, National Institute of Standards and Technology, Gaithersburg, MD.

Barika, F., Hadjar, K., and El-Kadhi, N. (2009). Artificial neural network for mobile ids solution. *Security and Management*, pages 271–277.

Barman, D. K. and Khataniar, G. (2012). Design of intrusion detection system based on artificial neural network and application of rough set. *International Journal of Computer Science and Communication Networks*, 2(4):548–552.

Bergadano, F., Gunetti, D., and Picardi, C. (2003). Identity verification through dynamic keystroke analysis. *Intell. Data Anal.*, 7(5):469–496.

Berman, D., Buczak, A., Chavis, J., Corbett, C., Berman, D. S., Buczak, A. L., Chavis, J. S., and Corbett, C. L. (2019). A Survey of Deep Learning Methods for Cyber Security. *Information*, 10(4):122.

BrainJS (2016). brain.js.

Bridges, S. M., Vaughn, R. B., and Siraj, A. (2002). Ai techniques applied to high performance computing intrusion detection. In *Proceeding of the Tenth International Conference on Telecommunication Systems, Modeling and Analysis, Monterey CA*, volume 2, pages 100–114.

Bright Computing (2019). Bright Cluster Manager v9.

Brooke, J. (1995). Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189.

Buczak, A. L. and Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176.

Bulusu, R., Jain, P., Pawar, P., Afzal, M., and Wandhekar, S. (2018). Addressing security aspects for HPC infrastructure. In *2018 International Conference on Information and Computer Technologies, ICICT 2018*, pages 27–30. Institute of Electrical and Electronics Engineers Inc.

by Harald Welte (2000). The netfilter framework in Linux 2.4. Technical report.

Cadariu, M., Bouwers, E., Visser, J., and van Deursen, A. (2015). Tracking known security vulnerabilities in proprietary software systems. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 516–519. IEEE.

Caddy (2020). Caddy Web Server.

Calegari, P., Levrier, M., and Atos, F. (2019). Web Portals for High-performance Computing: A Survey. *ACM Trans. Web*, 13(5).

Campbell, S. and Mellander, J. (2011). Experiences with Intrusion Detection in High Performance Computing. Technical Report 9.

Center, O. S. (2020). ondemand. https://github.com/OSC/ondemand.

Chang, V., Li, C.-S., De Roure, D., Wills, G., Walters, R. J., and Chee, C. (2011). The Financial Clouds Review. *International Journal of Cloud Applications and Computing*, 1(2):41–63.

Chen, Y., Zheng, Q., and Yang, H. (2017). A kind of university HPC platform security balance method based on the barrel theory. In *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, pages 3708–3713. Institute of Electrical and Electronics Engineers Inc.

Chugh, U. and Chugh, A. (2010). Security in high performance computing. In *Information Processing and Management*, pages 552–556, Berlin, Heidelberg. Springer Berlin Heidelberg.

Clough, J. (2015). *Principles of Cybercrime*. Cambridge University Press, Cambridge.

Clusterworks (2018). Clusterworks - Inception.

Cobb, C., Sudar, S., Reiter, N., Anderson, R., Roesner, F., and Kohno, T. (2018). Computer security for data collection technologies. *Development Engineering*, 3:1–11.

Corp Pei Symantec J Rydell, D. M. (2011). Internet Engineering Task Force (IETF) TOTP: Time-Based One-Time Password Algorithm.

Cuppens, M. (2016). Implementation and Analysis of a ChaCha Hardware Accelerator for the Caddy Secure Webserver Mathijs Cuppens. Technical report.

CyVerse (2020). Science APIs | CyVerse.

Daniel J. Barrett, Richard E. Silverman, R. G. B. (2005). SSH, The Secure Shell: The Definitive Guide: The Definitive Guide.

Dasgupta, D. (2006). Computational intelligence in cyber security. In *2006 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, pages 2–3.

Despa, M. L. (2014). Comparative study on software development methodologies. *Database Systems Journal*, 5(3):37–56.

Dewayne Adams (2011). Six security risks in High Performance Computing (HPC).

Dheshmukh, A. T. and Mahalle, P. N. (2014). Survey on Linux Security and Vulnerabilities. Technical report.

Dilek, S., Cakır, H., and Aydın, M. (2015). Applications of artificial intelligence techniques to combating cyber crimes: A review. *International Journal of Artificial Intelligence & Applications*, 6(1):21–39.

Donevski, M. and Zia, T. (2018). A Survey of Anomaly and Automation from a Cybersecurity Perspective. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE.

Durumeric, Z., Ma, Z., Springall, D., Barnes, R., Sullivan, N., Bursztein, E., Bailey, M., Halderman, J. A., and Paxson, V. (2017). The Security Impact of HTTPS Interception.

EGI CSIRT (2020). Academic data centers abused for crypto currency mining . Technical report.

Eldred, M., Adams, C., and Good, A. (2015). Impact of EU data protection laws on cloud computing: Capturing cloud-computing challenges and fault lines. In *Delivery and Adoption of Cloud Computing Services in Contemporary Organizations*, pages 56–79. IGI Global.

Eldred, M., Good, A., and Adams, C. (2016). A case study on data protection and security decisions in cloud HPC. In *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*, pages 564–568. Institute of Electrical and Electronics Engineers Inc.

Ertaul, L., Kaur, M., and Gudise, V. A. K. R. (2016). Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms. In *Proceedings of the International Conference on Wireless Networks (ICWN)*, page 66. The Steering Committee of The World Congress in Computer Science, Computer ....

EshghiShargh, A. (2009). Using artificial immune system on implementation of intrusion detection systems. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, pages 164–168. IEEE.

Fenton, N. (2014). *Software metrics : a rigorous and practical approach*. CRC Press, Boca Raton, FL.

Fenzi, K. and Wreski, D. (2000). Linux Security HOWTO. Technical report.

Ferreira, E. W. T., Carrijo, G. A., de Oliveira, R., and de Souza Araujo, N. V. (2011). Intrusion detection system with wavelet and neural artifical network approach for networks computers. *IEEE Latin America Transactions*, 9(5):832–837.

Foster, I., Karonis, N., Kesselman, C., and Tuecke, S. (1998). Managing security in high-performance distributed computations. *Cluster Computing*, 1(0):95–107.

Fridman, A., Stolerman, A., Acharya, S., Brennan, P., Juola, P., Greenstadt, R., and Kam, M. (2013). Decision fusion for multimodal active authentication. *IT Professional*, 15(4):29–33.

Fridman, L., Weber, S., Greenstadt, R., and Kam, M. (2017). Active authentication on mobile devices via stylometry, application usage, web browsing, and gps location. *IEEE Systems Journal*, 11(2):513–521.

Fujitsu (2010). SynfiniWay V4.

Fujitsu (2015). HPC Workload-optimized Solutions : FTS : HPC - HPC Workload-optimized Solutions : Fujitsu Global.

Gantikow, H., Reich, C., Knahl, M., and Clarke, N. (2016). Providing security in container-based HPC runtime environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9945 LNCS, pages 685–695. Springer Verlag.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28.

Greenberg, J. P., Mock, S., Bhatia, K., Katz, M., Bruno, G., Sacerdoti, F., Papadopoulos, P., and Baldridge, K. K. (2005). Incorporation of middleware and grid technologies to enhance usability in computational chemistry applications. volume 21, pages 3–10. North-Holland.

Hassan, M., Ali, M., Bhuiyan, T., Sharif, M., and Biswas, S. (2018). Quantitative assessment on broken access control vulnerability in web applications.

Hassan, M. M. M. (2013). Network intrusion detection system using genetic algorithm and fuzzy logic.

Herrero, Á., Corchado, E., Pellicer, M. A., and Abraham, A. (2007). Hybrid multi agent-neural network intrusion detection with mobile visualization. In *Innovations in Hybrid Intelligent Systems*, pages 320–328. Springer.

Higgins, J., Al-Jody, T., and Holmes, V. (2018). Rapid Deployment of Bare-Metal and In-Container HPC clusters using OpenHPC playbooks. Technical report.

Higgins, J., Holmes, V., and Venters, C. (2016). Securing user defined containers for scientific computing. In *2016 International Conference on High Performance Computing and Simulation, HPCS 2016*, pages 449–453. Institute of Electrical and Electronics Engineers Inc.

Hoffman, D., Prabhakar, D., and Strooper, P. (2003). Testing iptables. In *Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '03, page 80–91. IBM Press.

HPCsec (2019). Do Traditional Cyber Security Rules Apply to HPC Environments? - HPCSec.

Hudak, D., Johnson, D., Chalker, A., Nicklas, J., Franz, E., Dockendorf, T., and L. McMichael, B. (2018). Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software*, 3(25):622.

Hudak, D. E., Johnson, D., Nicklas, J., Franz, E., McMichael, B., and Gohar, B. (2016). Open OnDemand: Transforming computational science through omnidisciplinary software cyberinfrastructure. In *ACM International Conference Proceeding Series*, volume 17-21-July-2016. Association for Computing Machinery.

Hudak, D. E., Johnson, D., Nicklas, J., Franz, E., McMichael, B., and Gohar, B. (2017). OSC/Open-OnDemand: Supercomputing. Seamlessly. Open, Interactive HPC Via the Web.

Inside HPC (2015). Fujitsu Launches HPC Gateway Web Software .

James D. Ballew, Shannon V. Davidson, and Anthony N. Richoux (2015). System and method for cluster management based on HPC architecture.

Jankowicz, A. (2000). *Business Research Projects*. Thomson Learning.

Janssen, R. (2020). Twofactorauth. https://github.com/RobThree/TwoFactorAuth.

Jon Oltsik (2018). The evolution of security operations, automation and orchestration | CSO Online.

Jongsuebsuk, P., Wattanapongsakorn, N., and Charnsripinyo, C. (2013). Real-time intrusion detection with fuzzy genetic algorithm. In *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–6. IEEE.

Kadlecsik, J., Rmki, K., Pásztor, G., and Ek, S. (2004). Netfilter Performance Testing. Technical report.

Kanclirz, J. and Baskin, B. (2008). *Netcat power tools*. Syngress Pub.

Kesteren, M. (2019). A New Dawn for Security Vulnerabilities in HPC.

Khajeh-Hosseini, A., Sommerville, I., and Sriram, I. (2010). Research challenges for enterprise cloud computing.

Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2018). Spectre Attacks: Exploiting Speculative Execution. Technical report.

Kogos, K. G., Finoshin, M. A., and Gentyuk, V. A. (2019). Internet users authentication via artificial intelligence. In *Biologically Inspired Cognitive Architectures Meeting*, pages 219–224. Springer.

Köhler, J., Simon, M., Nussbaumer, M., and Hartenstein, H. (2013). Federating HPC access via SAML: Towards a plug-and-play solution. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7905 LNCS, pages 462–473. Springer, Berlin, Heidelberg.

Kruegel, C. (2004). Network security and secure applications.

Kuhn, M. (2005). UTF-8 and Unicode FAQ for Unix/Linux. Technical report.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Linda, O., Vollmer, T., and Manic, M. (2009). Neural network based intrusion detection system for critical infrastructures. In *2009 international joint conference on neural networks*, pages 1827–1834. IEEE.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. (2018). Meltdown: Reading Kernel Memory from User Space. Technical report.

Luo, Z., Qu, Z., Nguyen, T. T., Zeng, H., and Lu, Z. (2019). Security of HPC Systems: From a Log-analyzing Perspective.

Malin, A. and Van Heule, G. (2013). Continuous monitoring and cyber security for high performance computing. In *CLHS 2013 - Proceedings of the ACM Workshop on Changing Landscapes in HPC Security*, pages 9–14, New York, New York, USA. ACM Press.

Marquardson, J. (2012). Password Policy Effects on Entropy and Recall: Research in Progress. *AMCIS 2012 Proceedings*.

Marru, S., Gunathilake, L., Herath, C., Tangchaisin, P., Pierce, M., Mattmann, C., Singh, R., Gunarathne, T., Chinthaka, E., Gardler, R., Slominski, A., Douma, A., Perera, S., and Weerawarana, S. (2011). Apache Airavata: A framework for distributed applications and computational workflows. In *GCE'11 - Proceedings of the 2011 ACM Workshop on Gateway Computing Environments, Co-located with SC'11*, pages 21–28, New York, New York, USA. ACM Press.

McCormack, R. P., Koontz, J. E., and Devaney, J. (1999). Seamless computing with WebSubmit. *Concurrency: Practice and Experience*, 11(15):949–963.

McCormick, M. (2012). Waterfall vs. agile methodology. *MPCS, N/A*.

McGraw, G. (2006). *Software security : building security in*. Addison-Wesley, Upper Saddle River, NJ.

Meister, B. (2007). PuTTY/Cygwin Tutorial. Technical report.

Mendes, N., Neto, A. A., Duräes, J., Vieira, M., and Madeira, H. (2008). Assessing and comparing security of web servers. In *Proceedings of the 14th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2008*, pages 313–322.

Metla, H., Vinay Reddy, M., Sai Kiran, C., and Madhu Viswanatham, V. (2018). Crypt-analysis of secure hash password technique (cshpt) in linux. In *2018 IADS International Conference on Computing, Communications & Data Engineering (CCODE)*, pages 7–8.

MITRE (2019). CVE - CVE-2019-14287.

Morris, R. and Thompson, K. (1979). Password security: A case history. *Commun. ACM*, 22(11):594–597.

Newham, C. (2005). *Learning the bash Shell: Unix Shell Programming*. "O'Reilly Media, Inc.".

Nguyen, T. T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76.

Nicolett, M. and Kavanagh, K. M. (2011). Magic Quadrant for Security Information and Event Management Evaluation Criteria Definitions.

Nimbix (2012). JARVICE is the Cloud Platform for Big Compute.

NIST (1998). WebSubmit Home Page; remote access via the web; WebSubmit; websubmit.

Nogueira, J. (2006). Mobile intelligent agents to fight cyber intrusions. *The International Journal of Forensic Computer Science*, pages 28–32.

OpenHPC Community (2018). OpenHPC.

Ou, C.-M., Wang, Y.-T., and Ou, C.-R. (2011). Intrusion detection systems adapted from agent-based artificial immune systems. In *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, pages 115–122. IEEE.

OWASP (2017). A6:2017-Security Misconfiguration | OWASP.

OWASP.org (2017). OWASP Top Ten 2017.

Patel, A., Taghavi, M., Bakhtiyari, K., and Júnior, J. C. (2012). Taxonomy and proposed architecture of intrusion detection and prevention systems for cloud computing. In *International Symposium on Cyberspace Safety and Security*, pages 441–458. Springer.

Peisert, S. (2017). Security in high-performance computing environments.

Peisert, S. (2018). Cybersecurity for hpc systems: State of the art and looking to the future. Technical report.

Peisert-Lawrence Berkeley National Lab, S. (2016). HPC Workshop Presentation - Cybersecurity for HPC Systems: Challenges and Opportunities.

Pellerin David, Ballantyne Dougal, and Boeglin Adam (2015). An Introduction to High Performance Computing on AWS. Technical report.

Petersen, K., Wohlin, C., and Baca, D. (2009). The waterfall model in large-scale development. *Lecture Notes in Business Information Processing Product-Focused Software Process Improvement*, page 386–400.

PHPAuth (2020). Phpauth. https://github.com/PHPAuth/PHPAuth.

Pierce, M. E., Marru, S., Gunathilake, L., Kanewala, T. A., Singh, R., Wijeratne, S., Wimalasena, C., Herath, C., Chinthaka, E., Mattmann, C., Slominski, A., and Tangchaisin, P. (2014). Apache Airavata: Design and directions of a science gateway framework. In *Proceedings - 6th International Workshop on Science Gateways, IWSG 2014*, pages 48–54. IEEE Computer Society.

Platform Computing (2009). Platform Application Center introduced by Platform Computing. - Free Online Library.

Ponemon (2013). The Post Breach Boom Sponsored by Solera Networks. Technical report.

Ponemon Institute (2017). 2017 Cost of Data Breach Study Global Overview. Technical report.

Priedhorsky, R. and Randles, T. (2017). Charliecloud: Unprivileged containers for user-defined sofware stacks in HPC. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017*, pages 1–10, New York, New York, USA. Association for Computing Machinery, Inc.

Prout, A., Arcand, W., Bestor, D., Bergeron, B., Byun, C., Gadepally, V., Hubbell, M., Houle, M., Jones, M., Michaleas, P., Milechin, L., Mullen, J., Rosa, A., Samsi, S., Reuther, A., and Kepner, J. (2016). Enhancing HPC security with a user-based firewall. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016*. Institute of Electrical and Electronics Engineers Inc.

Provos, N. and Mazieres, D. (1999). Bcrypt algorithm. In *USENIX*.

Purdy, G. N. (2004). *Linux iptables : pocket reference*. O'Reilly Media.

Purgason, B. and Hibler, D. (2012). Security through behavioral biometrics and artificial intelligence. *Procedia Computer Science*, 12:398–403.

Quintero, D. and International Business Machines Corporation. International Technical Support Organization. (2012). *IBM Platform Computing solutions*. IBM Corp., International Technical Support Organization.

Rasheed, H. (2014). Data and infrastructure security auditing in cloud computing environments. *International Journal of Information Management*, 34(3):364–368.

Red Hat (2016). Dirty Copy-On-Write.

Red Hat (2017). Speculative Execution Exploit Performance Impacts - Describing the performance impacts to security patches for CVE-2017-5754 CVE-2017-5753 and CVE-2017-5715 - Red Hat Customer Portal.

Red Hat (2019). Ansible is Simple IT Automation.

Reese, K., Smith, T., Dutson, J., Armknecht, J., Cameron, J., and Seamons, K. (2019). A usability study of five two-factor authentication methods. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, Santa Clara, CA. USENIX Association.

Rescale (2012). ScaleX .

Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. (1998). Virtual network computing.

Rocksclusters (2018). ROCKS OS.

Sadowski, C. and Shewmaker, A. (2010). The last mile: Parallel programming and usability. In *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010*, pages 309–313, New York, New York, USA. ACM Press.

Sampedro, Z., Hauser, T., and Sood, S. (2017). Sandstone HPC-A domain-general gateway for new HPC users. In *ACM International Conference Proceeding Series*, volume Part F128771, pages 1–7, New York, New York, USA. Association for Computing Machinery.

Serwadda, A., Phoha, V. V., and Kiremire, A. (2011). Using global knowledge of users' typing traits to attack keystroke biometrics templates. In *Proceedings of the Thirteenth ACM Multimedia Workshop on Multimedia and Security*, page 51–60, New York, NY, USA. Association for Computing Machinery.

Shannon, C. E. (1956). *Automata Studies.* Princeton University Press.

Shao-jing, Z., Wen-li, S., Hui, R., Zhen-ting, C., and Yu, Y. (2011). Research of intelligent immune intrusion detection system about combating virus with "virus". In *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, pages 753–756. IEEE.

Shosha, A. F., Gladyshev, P., Wu, S.-S., and Liu, C.-C. (2011). Detecting cyber intrusions in scada networks using multi-agent collaboration. In *2011 16th International conference on intelligent system applications to power systems*, pages 1–7. IEEE.

Simakov, N. A., Innus, M. D., Jones, M. D., White, J. P., Gallo, S. M., DeLeon, R. L., and Furlani, T. R. (2018). Effect of Meltdown and Spectre Patches on the Performance of HPC Applications.

Sims, J. S., Hagedorn, J. G., Ketcham, P. M., Satterfield, S. G., Griffin, T. J., George, W. L., Fowler, H. A., Am Ende, B. A., Hung, H. K., Bohn, R. B., Koontz, J. E., Martys, N. S., Bouldin, C. E., Warren, J. A., Feder, D. L., Clark, C. W., Filla, B. J., and Devaney, J. E. (2001). Accelerating Scientific Discovery through Computation and Visualization. Technical report.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., and Stiller, B. (2010). An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356.

Stan Engelbrecht (2018). The Evolution of SOAR Platforms | SecurityWeek.Com.

Swift, D. (2006). A Practical Application of SIM/SEM/SIEM Automating Threat Identification.

SysFera (2014). SysFera-DS Version 5.0 Introduced.

Teixeira, P. (2012). *Professional Node.js : building Javascript based scalable software*. Wiley.

The Agave Platform (2013). The Agave Platform.

Tyugu, E. (2011). Artificial intelligence in cyber defense. In *2011 3rd International Conference on Cyber Conflict*, pages 1–11.

UNICORE (1997). UNICORE | Distributed computing and data resources.

Univa (2019). Grid Engine.

Vaarandi, R. and Pihelgas, M. (2014). Using security logs for collecting and reporting technical security metrics. In *Proceedings - IEEE Military Communications Conference MILCOM*, pages 294–299. Institute of Electrical and Electronics Engineers Inc.

Vacca, J. R. (2004). *Public key infrastructure : building trusted applications and Web services*. Auerbach Publications.

Verizon (2019). 2019 Data Breach Investigations Report | Verizon Enterprise Solutions.

Verizon (2020). Verizon: Data breach investigations report 2020. *Computer Fraud and Security*, 2020(6):4.

Viega, J. (2002). *Building secure software : how to avoid security problems the right way*. Addison-Wesley, Boston.

Vue.js (2020). Vue.js.

We Are Social (2020). Digital 2020 - We Are Social.

Wheeler, D. L. (2016). zxcvbn: Low-budget password strength estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX. USENIX Association.

Wickramasinghe, C., Marino, D., Amarasinghe, K., and Manic, M. (2018). Generalization of Deep Learning for Cyber-Physical System Security: A Survey. pages 745–751.

Wu, S. X. and Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013*, pages 233–240.

XCAT (2018). xCAT | Extreme Cloud Administration Toolkit.

Xiao-bin Wang, Guang-yuan Yang, Yi-chao Li, and Dan Liu (2008). Review on the application of artificial intelligence in antivirus detection systemi. In *2008 IEEE Conference on Cybernetics and Intelligent Systems*, pages 506–509.

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., and Wang, C. (2018). Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access*, 6:35365–35381.

Zhang, Y., Monrose, F., and Reiter, M. K. (2010). The security of modern password expiration: An algorithmic framework and empirical analysis. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 176–186, New York, New York, USA. ACM Press.

Zhang, Y., Wang, L., Sun, W., Green, R. C., and Alam, M. (2011). Artificial immune system based intrusion detection in a distributed hierarchical network architecture of smart grid. In *2011 IEEE Power and Energy Society General Meeting*, pages 1–8. IEEE.

Appendix A

Engagement with The Research Community

Publications

Al-Jody, T., Holmes, V., Antoniades, A., & Kazkouzeh, Y.: Bearicade: secure access gateway to High Performance Computing systems. In The 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications 2020.

Abstract: *"Cyber security is becoming a vital part of many information technologies and computing systems. Increasingly, High-Performance Computing systems are used in scientific research, academia and industry. High-Performance Computing applications are specifically designed to take advantage of the parallel nature of High-Performance Computing systems. Current research into High-Performance Computing systems focuses on the improvements in software development, parallel algorithms and computer systems architecture. However, there are no significant efforts in developing common High-Performance Computing security standards. Security of the High-Performance Computing resources is often an add-on to existing varied institutional policies that do not take into account additional requirements*

for High-Performance Computing security. Also, the users' terminals or portals used to access the High-Performance Computing resources are frequently insecure or they are being used in unprotected networks. In this paper we present Bearicade - a Data-driven Security Orchestration Automation and Response system. Bearicade collects data from the HPC systems and its users, enabling the use of Machine Learning based solutions to address current security issues in the High-Performance Computing systems. The system security is achieved through monitoring, analysis and interpretation of data such as users' activity, server requests, devices used and geographic locations. Any anomaly in users' behaviour is detected using machine learning algorithms, and would be visible to system administrators to help mediate the threats. The system was tested on a university campus grid system by administrators and users. Two case studies, Anomaly detection of user behaviour and Classification of Malicious Linux Terminal Command, have demonstrated machine learning approaches in identifying potential security threats. Bearicade's data was used in the experiments. The results demonstrated that detailed information is provided to the HPC administrators to detect possible security attacks and to act promptly."

Higgins, J, Al-Jody, T, Aagela, H., Holmes, V.: "Inspiring the Next Generation of HPC Engineers with Re-configurable, Multi-Tenant Resources for Teaching and Research", Under Review.

Abstract: *"There is a tradition at our university for teaching and research in High Performance Computing (HPC) systems engineering. With exascale computing on the horizon, and a shortage of HPC talent, there is a need for new research computing specialists to secure the future of research computing. Whilst many institutions provide research computing training for users within their particular domain, few offer HPC engineering and infrastructure related courses, making it difficult for students to acquire these skills. This paper outlines how and why we are training students in HPC systems engineering skills, including technologies used in delivering this goal. We demonstrate a potential for a multi-tenant system for education and research which can be supported by other institutions, using novel container and cloud*

based architecture. An evaluation of our activities over the last 2 years is given in terms of recruitment metrics, skills audit feedback from students, and research outputs enabled by the multi-tenant usage of the resource.".

Higgins, J., Al-Jody, T. & Holmes, V.: "Rapid Deployment of Bare-Metal and In-Container HPC clusters using OpenHPC playbooks", HPC Systems Professionals Workshop, held in conjunction with SuperComputing 2018.

Abstract: *"In this paper, we present a toolbox of reusable Ansible roles and playbooks in order to configure a cluster software environment described by the freely available OpenHPC recipes. They can be composed in order to deploy a robust and reliable cluster environment, instilled with the best practise offered by the OpenHPC packaging, and the repeatability and integrity guarantees of the configuration managed approach. With container virtualization setting a new trend in scientific software management, we focus this effort on supporting the deployment of such environments on both bare-metal and container-based targets using the Virtual Container Cluster framework."*

Aagela, H., Al-Jody, T., Holmes, V.: "Web-based Wireless Wake-on-LAN approach for Robots.", IEEE, ICAC 2018.

Abstract: *"The ability to remotely wake-up robots over a wireless LAN can improve the performance and the power consumption of remote robots or cloud robotics system. This paper presents a solution for power management of a mobile robot, using web-based wireless Wake-on-LAN (WWoL). The focus in on power management of a mobile robot, but this approach is also suitable for most of the IoT mobile devices, and other systems that are designed to be used remotely. The proposed solution allows the targeted device to be powered ON and OFF remotely via a web-based dashboard. This approach is validated in a case study with an AR. Drone 2.0. and demonstrated substantial power optimisation for the drone. In addition, security issues are explored when WWoL is deployed in remote control of mobile*

devices. It was established that a power consumption is reduced when the drone is in a standby mode waiting for an operator to send a wake-on request message wirelessly, without compromising security posed by wireless remote access to the devices".

Matani, A., Al-Jody, T. & Benson, D.: "Pilot Investigating the Correlation between Attainment and Attendance using Statistics, Neural Networking and Artificial Intelligence", JALTIE Journal 2019.

Abstract: *"This paper describes a new approach in quantitatively predicting students' performance using Artificial Intelligence and Neural Networks (AI&NN) systems. The purpose of the study is to investigate the suitability of these systems to accurately predict students' attainment, from us- ing factors of attendance and pre-entry English scores. One-year's worth of anonymous data of yearly attendance percentages, past pre-entry English scores, and actual attainment was used from 2017-18 published annual evaluation reports data-set and attendance records for Hudders- field International Study Centre, UK. The study used this data to train the AI&NN system, and the latter, in turn, learnt to predict attainment of similar arbitrary data. The AI&NN system also succeeded in quantitatively predicting, after it learnt from the real data, the significant influence of attendance on attainment compared to the influence of the other factors used. This method is chosen over other methods (Costa et al., 2017; Kumar et al., 2017) because it is powerful; its outcome is fast and scalable. The study's encouraging results show that, if trained with the right amount of data and used in the right context, similar systems will have the potential to be used to predict many other desired outcomes that will have a great impact on improving the quality of education."*

Al-Jody, T., Holmes, V.: "Security Orchestration Automation and Response (SOAR) in High Performance Computing Systems", Emerging Technology Conference 2019.

Abstract: *"Cybersecurity is becoming critical for many information and computing systems. In addition to standard security issues, there are additional requirements for secure access*

to High-Performance Computing (HPC) systems. The user management of HPC systems is often an add-on to existing institutional policies that do not take into account the need to access many different HPC systems. There are no common HPC security standards in research and education communities. Every institution has its own policy, often certified by the e-science certificate authority (CA). Because the HPC systems are often behind institutional firewalls, most registered users use Simply Secure Shell (SSH) to access given HPC resource. The institutions often hold training events to educate the users on how to access their HPC resources. This is sometimes perceived as a barrier to entry to HPC. However, the systems used to access HPC clusters are frequently insecure, such as Putty, or they are being used in unprotected networks. Currently, many HPC centres make use of Private Key Infrastructure (PKI) as a basic method of authentication. This method has proven reliability. However, even with the PKI authentication in place, there are potentially security issues for users who have minimal security awareness. In this paper we present a Security Orchestration Automation and Response (SOAR) system which will address current security issues in HPC systems. It is based on familiar browser elements and supports secure operation and access to HPC systems without a need for specialist user training. The architecture of the systems is shown in figure 1. The user registration and authorisation is based on the two-factor authentication, via either the time-based one-time password or universal second factor (U2F). The system security is achieved through monitoring, analysis and interpretation of data such as continuous supervision of users' activity, server requests, devices and geographic locations, etc. Any anomaly in users' behavior would be visible to system administrators and will facilitate prompt actions in order to mediate the threats. The system was tested on Queens Gate Grid (QGG) system by administrators and student users at the University of Huddersfield. Initial administrator and user feedback is encouraging. From an administrative perspective, detailed information is available to detect possible security threats and to act promptly. From the users' perspective, the system is easy to learn and use. It provides repeatedly reliable access to resources, submitting and running jobs, and managing files.“

Open Source Projects

Bearicade: *"Bearicade is an MIT-licensed open-source data-driven secure gateway for distributed system, build on a REST API, containerized via Docker and deployable with Ansible."*

Github Repository | Website

Web-User-Behaviour: *"Configurable and Lightweight JS Library for user behaviour tracking from the browser, using mouse movements, clicks, scroll, and time on page."*

Github Repository

Inception: *"Inception is a toolkit that brings together the best modern technologies in order to create fast and flexible turn-key HPC environments, deployable on bare-metal infrastructure or in the cloud."*

Github Repository

Appendix B

Surveys Template

The System Usability Scale (SUS) surveys issued to the students as explained in 4.5 are shown in figures B.1 and B.2 respectively.

Please enter your student number: U_____

System Usability Scale (SUS)

This is a standard questionnaire that measures the overall usability of a system that you have built in term 1. Please select the answer that best expresses how you feel about each statement after using your cluster.

		Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
1.	I think I would like to use this tool frequently.	□	□	□	□	□
2.	I found the tool unnecessarily complex.	□	□	□	□	□
3.	I thought the tool was easy to use.	□	□	□	□	□
4.	I think that I would need the support of a technical person to be able to use this system.	□	□	□	□	□
5.	I found the various functions in this tool were well integrated.	□	□	□	□	□
6.	I thought there was too much inconsistency in this tool.	□	□	□	□	□
7.	I would imagine that most people would learn to use this tool very quickly.	□	□	□	□	□
8.	I found the tool very cumbersome to use.	□	□	□	□	□
9.	I felt very confident using the tool.	□	□	□	□	□
10.	I needed to learn a lot of things before I could get going with this tool.	□	□	□	□	□

How likely are you to recommend this cluster to others? (please circle your answer)

Not at all likely 0 1 2 3 4 5 6 7 8 9 10 Extremely likely

Fig. B.1 In-class SUS Survey for SSH access method

Compared to the first survey you have done about your experience in building the cluster, this survey is about the new Dashboard that you have used.

Question 1 (Mandatory)

I think that I would like to use this website frequently.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 2 (Mandatory)

I found this website unnecessarily complex.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 3 (Mandatory)

I thought this website was easy to use

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 4 (Mandatory)

I think that I would need assistance to be able to use this website.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 5 (Mandatory)

I found the various functions in this website were well integrated.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 6 (Mandatory)

I thought there was too much inconsistency in this website.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 7 (Mandatory)

I would imagine that most people would learn to use this website very quickly.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 8 (Mandatory)

I found this website very cumbersome/awkward to use.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 9 (Mandatory)

I felt very confident using this website

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 10 (Mandatory)

I needed to learn a lot of things before I could get going with this website.

◯Strongly Agree◯Agree◯Neutral◯Disagree◯Strongly Disagree

Question 11

Feedback or comments (optional)

Fig. B.2 Online SUS Survey for SSH access method

Appendix C

System Usability Scale (SUS) Results

The System Usability Scale (SUS) results explained in 4.5 are shown in Tables C.1 and C.2 for the SSH and Bearicade access method respectively.

Table C.1 System Usability Scale Results for SSH

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
1	4	3	4	4	4	4	3	2	4	2	60.0
2	4	3	4	5	5	2	4	3	5	2	67.5
3	3	1	3	5	3	2	4	3	5	3	60.0
4	3	2	4	4	3	3	4	4	2	4	47.5
5	4	3	2	5	4	2	2	3	3	4	45.0
6	5	2	5	4	3	3	5	3	5	4	67.5
7	3	3	3	3	3	3	3	3	3	3	50.0
8	2	3	4	4	4	2	3	1	3	2	60.0
9	2	3	3	4	3	3	2	3	3	5	37.5
10	4	4	2	3	3	2	2	4	3	5	40.0
11	4	4	3	5	5	5	3	4	3	3	42.5
12	3	4	2	4	4	3	2	3	2	4	37.5
13	3	3	2	3	3	3	3	3	4	5	45.0
14	3	3	3	3	3	3	3	3	3	5	45.0
15	3	3	3	4	4	2	4	3	2	4	50.0
16	2	3	2	2	2	4	2	1	3	3	45.0
17	3	2	4	1	4	1	5	4	4	1	77.5
18	4	2	5	2	4	2	5	5	5	3	72.5
19	5	2	5	2	5	1	5	2	5	3	87.5
20	5	1	3	2	5	3	5	5	5	2	75.0
21	3	2	4	3	5	2	5	3	4	3	70.0
22	1	5	1	5	4	4	2	3	2	5	20.0
23	3	2	3	3	3	3	3	3	3	3	52.5
24	3	3	3	3	3	3	3	3	3	3	50.0
25	5	4	4	4	5	1	2	1	4	4	65.0
26	3	2	3	2	4	4	3	3	3	4	52.5
27	4	3	3	2	4	4	2	2	3	2	57.5
28	4	3	4	2	4	4	4	3	4	2	65.0
29	4	2	3	4	4	3	4	2	4	4	60.0
30	2	2	4	3	5	4	2	2	3	2	57.5
31	4	5	3	3	3	3	3	3	3	3	47.5
32	1	2	5	4	4	2	4	3	3	3	57.5
33	4	3	3	2	2	3	4	3	4	4	55.0
34	5	4	4	4	5	4	4	5	4	5	50.0
35	3	1	4	3	4	2	4	2	3	5	62.5
36	2	4	3	5	3	3	2	3	1	5	27.5
37	2	3	2	5	3	4	1	1	1	5	27.5

Table C.2 System Usability Scale Results for Bearicade

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score
1	4	3	3	4	4	3	3	4	3	4	47.5
2	5	2	5	2	5	1	4	2	5	3	85.0
3	4	4	4	4	4	4	4	3	3	3	52.5
4	5	2	2	4	3	3	4	3	4	4	55.0
5	5	2	5	2	5	1	5	1	5	2	92.5
6	5	5	5	5	5	2	3	2	4	4	60.0
7	4	3	3	4	5	2	5	3	4	4	62.5
8	3	4	3	2	2	3	4	3	3	4	47.5
9	5	5	5	5	5	5	5	5	5	5	50.0
10	4	2	4	1	3	1	3	1	3	3	72.5
11	4	2	3	2	4	2	5	1	5	2	80.0
12	5	3	5	4	4	3	3	2	3	3	62.5

Appendix D

Bearicade: Summary and Source Code

Information about the source code and list of features for Bearicade could be found on the Website or the Github repository page:

Website	https://bearicade.ta3.dev
Source Code	https://github.com/TA3/bearicade

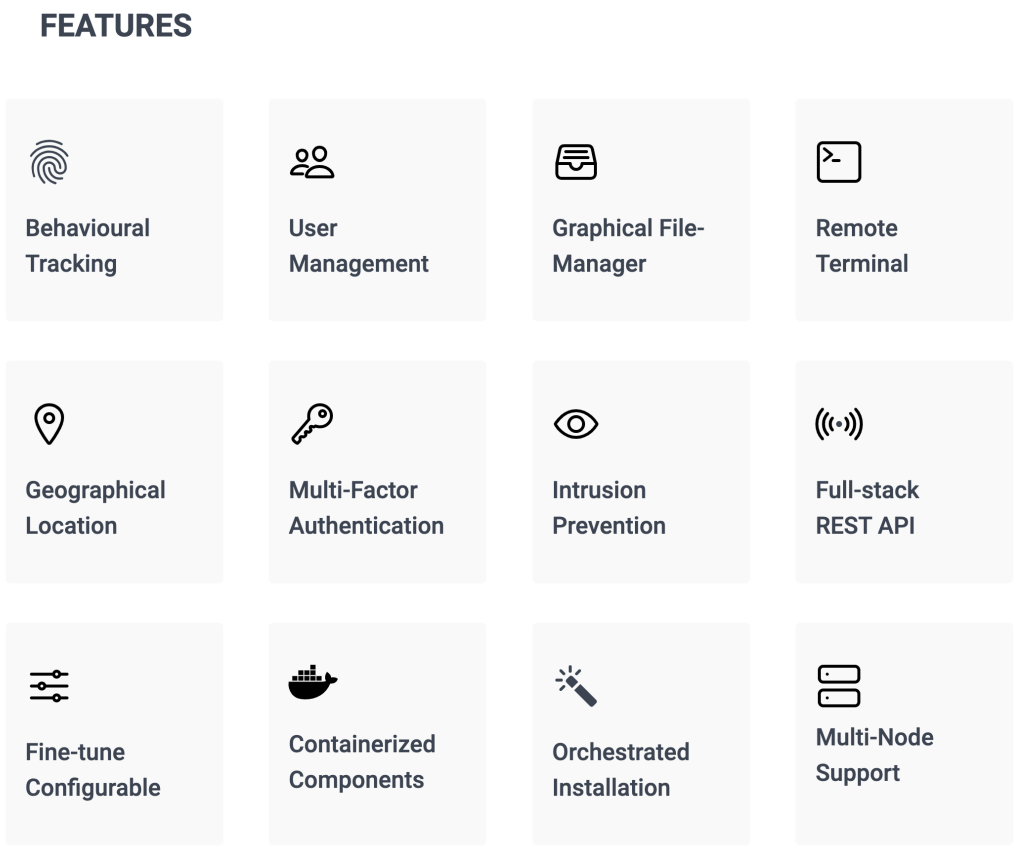

Fig. D.1 Some Features of Bearicade - Bearicade Website (Al-Jody, 2018)

Bearicade

version 0.1 documentation yes license MIT contributors 3

> Open-souce secure gateway for distributed system

Bearicade (pronounced `be(ə)ri,kād` from bear and ~~barr~~icade) is an MIT-licensed open-source data-drive secure gateway for distributed system, build on a REST API, containerized via Docker and deployable with Ansible.

Bearicade has been presented at the IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications Paper.

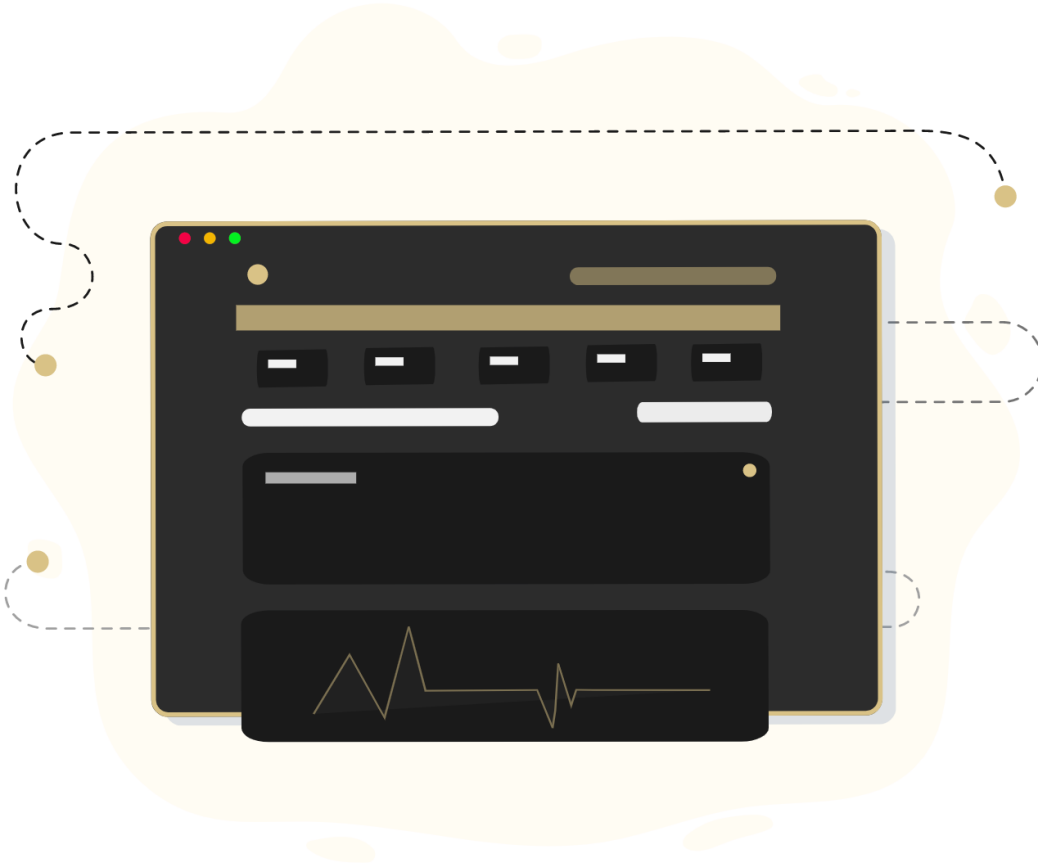

Fig. D.2 Summary of Bearicade - Bearicade Github Repository (Al-Jody, 2017)

Prerequisite

- CentOS/Red Hat >=7.4
- Git
- Python >=2.7
- PIP >= 20.0.2
- Ansible >= 2.9.4
- Docker >= 1.13.1

Deploy

1. Install Prerequisite

```
sudo yum install epel-release git python python-pip docker && sudo pip install ansible
```

2. Enable and Start Docker

```
sudo systemctl enable docker && systemctl start docker
```

3. Clone bearicade git repository

```
git clone https://github.com/TA3/bearicade
```

4. Generate Keys and Authorize the key

```
ssh-keygen && cat ~/.ssh/id_rsa.pub > ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys
```

5. Permit SSH Root Login

```
sed -i '/^PermitRootLogin/s/no/yes/' /etc/ssh/sshd_config && systemctl restart sshd
```

6. Edit config.yml to suit your preference

```
vi bearicade/bearicade_ansible/config.yml
```

7. Install bearicade with ansible playbook

```
ansible-playbook -i bearicade/bearicade_ansible/hosts bearicade/bearicade_ansible/bear_install.yml
```

8. Add Administrators with ansible playbook

```
ansible-playbook -i bearicade/bearicade_ansible/hosts bearicade/bearicade_ansible/bear_add_user.yml
```

Fig. D.3 Prerequisite and Deployment Procedure - Bearicade Github Repository (Al-Jody, 2017)