HEIMDALL: A DIGITAL THREAT MANAGEMENT PLATFORM

Functional Specification

by

Killian O'Connor

For IT Carlow

15/11/2019

Table of Contents

Table of Figures

1 General Description

The purpose of Heimdall is to provide a platform through which users can identify potential threats or information disclosures online, or vulnerabilities on their externally facing network, by utilising open source intelligence. The functionality of the platform will be achieved through different modules, focusing on unique use cases. Users will configure a data set of relevant keys. Users will also be able to prioritise these keys leading to a more granular approach.

The platform itself should have an API through which new modules can potentially be built, by leveraging new methods of intelligence collection or processing. The modular basis of the Heimdall platform will allow clients to customise their usage of modules to better suit their business needs.

2 Users

2.1 Admin

The admin user will be responsible for the administration of the site and accounts. This user will not have access to the modules but will have the functionality to manage client accounts. This user role will be held by an operator of the site, or somebody directly involved in the Heimdall Project. This user should be technically competent in both the general sense, and in regards to the inner workings of the Heimdall Project.

2.2 Client

A single client account will exist for an organisation. The account will utilise the functionality of the Heimdall Platform to gather and review threat intelligence. They will have access to the report generation module, and all the module functionalities that are currently enabled for the account, as well as have the option to enable or disable new modules. The client may add, delete, or update keys as well as the priority of the keys associated with their organisation. The account will be responsible for analysing and actioning alerts generated by the platform regarding their keys.

3 Context Diagram

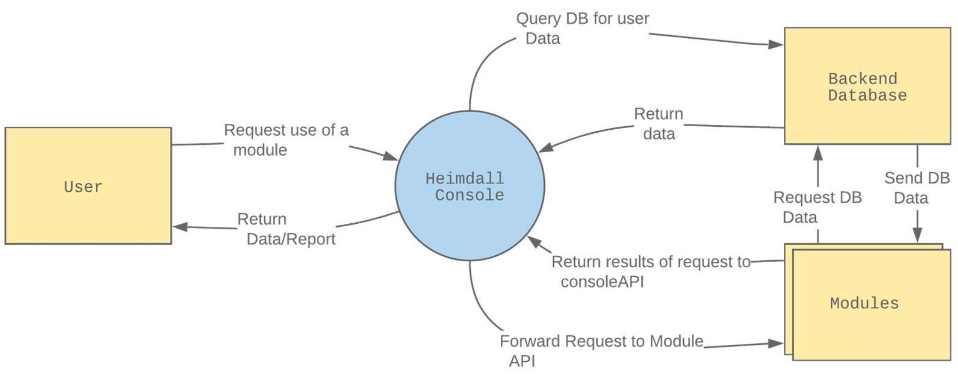

Figure 1 Context Diagram: Heimdall Platform

4 Use Cases

4.1 Heimdall Console

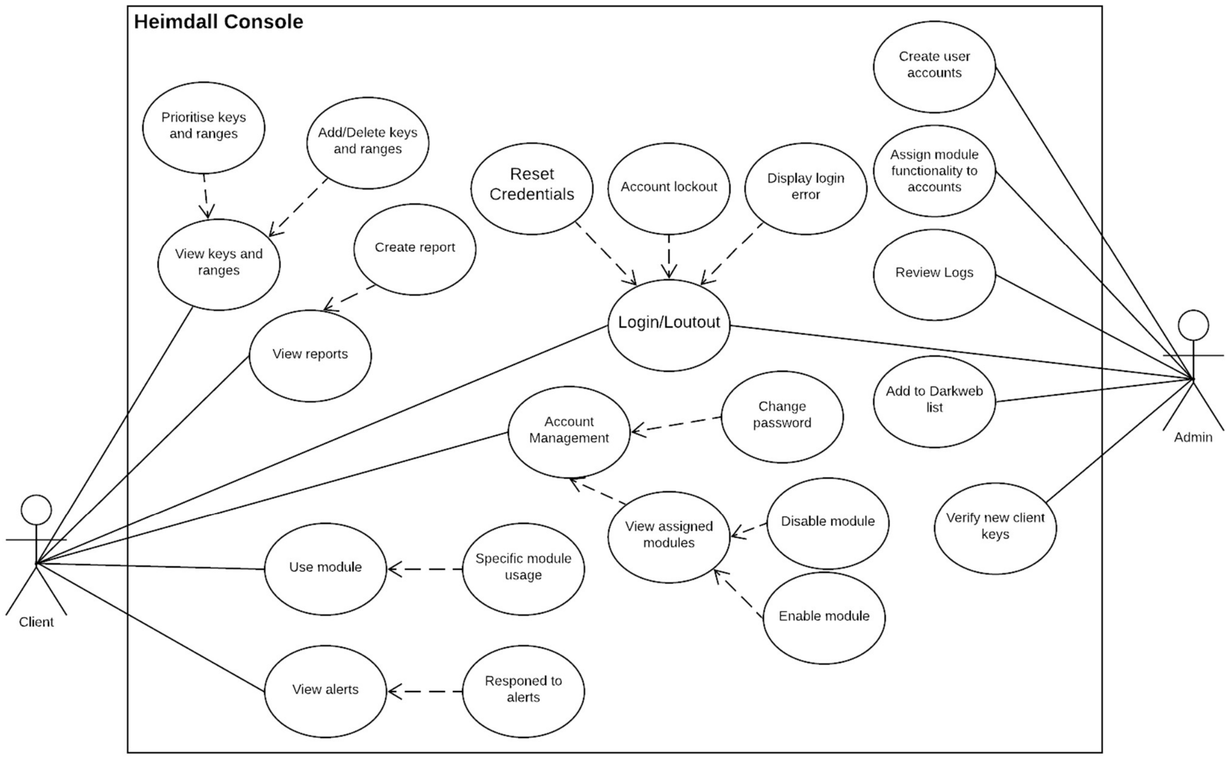

Figure 2 Use case: Heimdall Console

4.1.1 View Keys

Primary Actors:

Client.

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The relevant user keys are presented to the users.

Main Success Scenario:

1. The user selects the "Manage Keys" option.
2. The system will retrieve all keys with which the user account is associated.
3. The keys will populate the returned page in rows.
4. A user may set a priority value for keys.
5. A user may select individual keys to be deleted.
6. A user is presented with the option to add a key.

4.1.2 Create User Account

Primary Actors:

Admin.

Preconditions:

Actor initiating the use case is authenticated with the system and has admin privileges.

Success Guarantee:

A client account is created and accessible of the end user.

Main Success Scenario:

1. The admin account chooses the option "Create new client".

2. The admin supplies the username and password associated with the account.
3. The system creates the account and returns a success condition to the admin.

4.1.3 View Reports

Primary Actors:

Client.

Preconditions:

Actor initiating the use case is authenticated with the system.

Success Guarantee:

Users have the ability to download historical reports

Main Success Scenario:

1. The user selects the reporting functionality and is sent to the reporting home page.
2. The user is presented with the option to create a new report (3.1.6), as well as a list of previously generated reports.
3. The user many select and download a previously generated report.

4.1.4 Create Report

Primary Actors:

Client.

Preconditions:

Actor initiating the use case is authenticated with the system.

Success Guarantee:

System can successfully generate and return a report.

Main Success Scenario:

1. User select the “Create new report” option on the reporting home page.

2. Redirect to “Create Report” page.
3. The user will choose what report template to utilise.
4. The user will select the modules to include in the report.
5. The user will select a time period to utilise for report generation.
6. The user will submit the report request to the backend.
7. The system will pull all relevant alerts and populate a LaTeX template with the information.
8. The system will generate a pdf from the LaTeX file.
9. The system will return the pdf to the user.

4.1.5 View Alerts

Primary Actors:

Client

Preconditions:

Actor initiating the use case is authenticated with the system.

Success Guarantee:

The system generates a list of alerts for the user to individually action.

Main Success Scenario:

1. The user accesses the alerts page.
2. The system will retrieve all active alerts with which the user account is associated.
3. The alerts will populate the returned page in rows.
4. A user may select an alert to action and be brought to a page for the alert.
5. A user may add an “Analyst Comment” and/or “Remediate” the alert (Deactivate it).

4.1.6 Use Module

Primary Actors:

Client

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to utilise said module.

Success Guarantee:

System redirects and utilises the requested modules functionality.

Main Success Scenario:

1. User is presented with the option to access only the modules associated with their account.
2. The use will select the module they wish to use and proceed to that modules homepage.
3. The user will made use of said module (See further UML Diagrams below).

Side steps:

1.1 If the user has access to no modules an error notifying them to contact their administrator for access is returned.

4.1.7 Account Management

Primary Actors:

Admin, Client

Preconditions:

Actor initiating the use case is authenticated with the system.

Success Guarantee:

User is able to access and alter information associated with their account. E.g. email address, password, etc.

Main Success Scenario:

1. User selects the management option.
2. User is presented with information relating to their account and the functionality to change said information.
3. User changes a piece of information.
4. User selects the "Save Changes" button at the bottom of the screen.

5. The change request is sent to the server.
6. The request is authenticated and the information is changed.
7. The user is prompted to notify them whether the changes have been successful or not

Side steps:

3.1 The user attempts to change password.

3.2 The user enters their new password twice and their current password.

3.3 The request is authenticated and the information is changed.

3.4 The user is prompted to notify them whether the changes have been successful or not.

4.1.8 Login

Primary Actors:

Admin, Client

Preconditions:

Actor initiating the use case is unauthenticated.

Success Guarantee:

User either authenticates successfully or is blocked from the system after multiple failed login attempts.

Main Success Scenario:

1. User visits login page.
2. Submits enters credentials.
3. Username and password hash are passed to the server.
4. Server attempts to validate the credentials.
5. User credentials are valid.
6. User is authenticated with the system.
7. User is redirected to the authenticated home page.

Side steps:

5.1 User credentials are invalid.

5.2 User is prompted with generic invalid credential message.

5.3 If user has failed authentication multiple times previously, force a temporary lock out.

4.1.9 Logout

Primary Actors:

Admin, Client.

Preconditions:

Actor initiating the use case is authenticated with the system.

Success Guarantee:

User is deauthenticated with the system and must perform login again to access site functionality.

Main Success Scenario:

1. User selects “Logout” button.
2. Logout out request is sent to application.
3. User is deauthenticated.
4. User is sent to the home page of the web site.

4.2 Report Generation Module

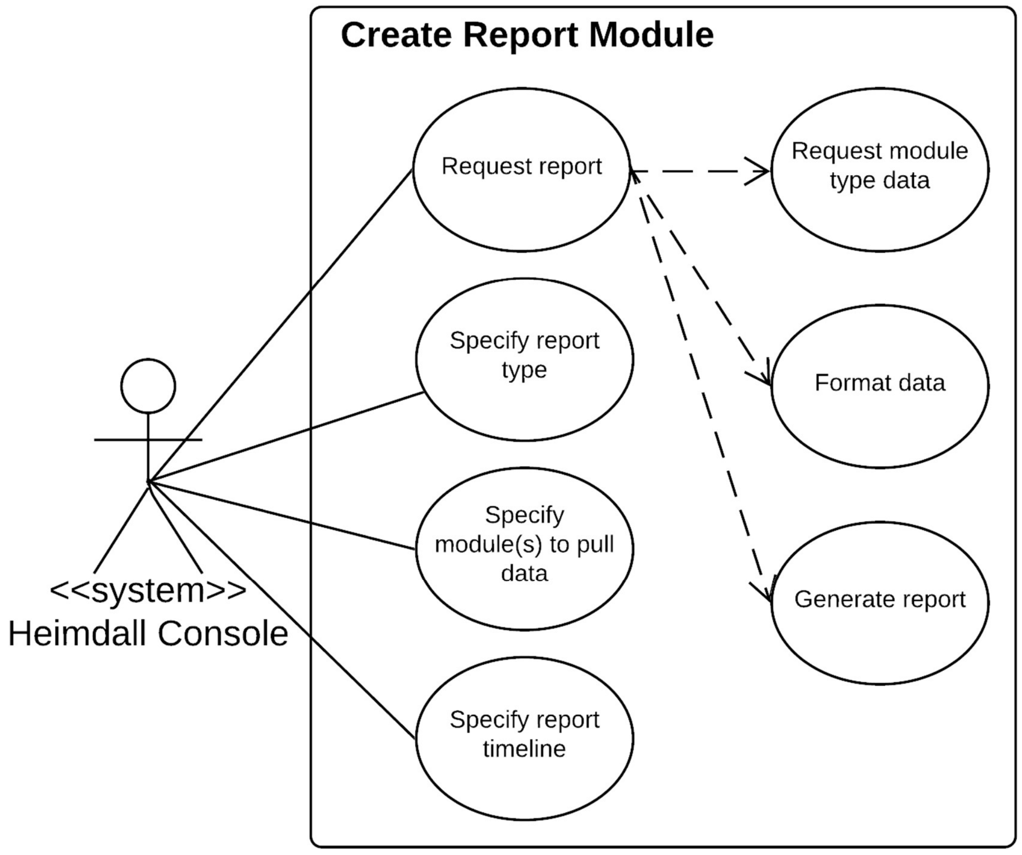

Figure 3 Use Case: Report Generation Module

4.2.1 Specify Report Type

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

User selects either "Technical" or "Executive" report format, and the next report generated will be done so using that format.

Main Success Scenario:

1. User selects the "Report Format" dropdown menu.
2. User selects either "Technical" or "Executive" option.

Side steps:

2.1 User selects no options and requests report.

2.2 System prompts user with error and cancels report generation

4.2.2 Specify Module(s) to Pull Data

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports. The user is only prompted with the option to select modules they have access to.

Success Guarantee:

The report generation function will only pull data the modules which the user selected, and generate the report on said data.

Main Success Scenario:

1. User is presented with radial selection of available modules.
2. User selects a subset of options

Side steps:

2.1. User selects no options and requests report.
2.2. System prompts user with error and cancels report generation

4.2.3 Specify Report Timeline

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The report generation function will only pull data present in the given timeline.

Main Success Scenario:

1. User sets the "From" date.
2. User sets the "To" date.

Side steps:

1.1. User does not set the "From" date and requests report.

1.2. System prompts user with error and cancels report generation

2.1. User does not set the "To" date and requests report.

2.2. The "To" date is assumed to be the current date.

4.2.4 Request Report

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports. The report format, timeline, and modules to pull data from have been set.

Success Guarantee:

The script executes completely, pulling metrics within the given timeframe from the database and formatting them in the chosen report format. The finished report should then be returned to the user as a downloadable pdf.

Main Success Scenario:

1. User issues report request.

2. System pulls entries from the database the match the specified modules and timeline.
3. System dynamically populates a LaTeX .tex file using either the Technical or Executive report template, whichever was specified in the request.
4. The .tex file is used to generate a pdf.
5. The report pdf is downloaded to the user's browser.

Side steps:

1.1. The request does not contain all required conditions to generate the report.
1.2. System prompts user with error and cancels report generation.
2.1. There is no corresponding user data.
2.2. System prompts user with error and cancels report generation.

4.3 AWS S3 Scanner module

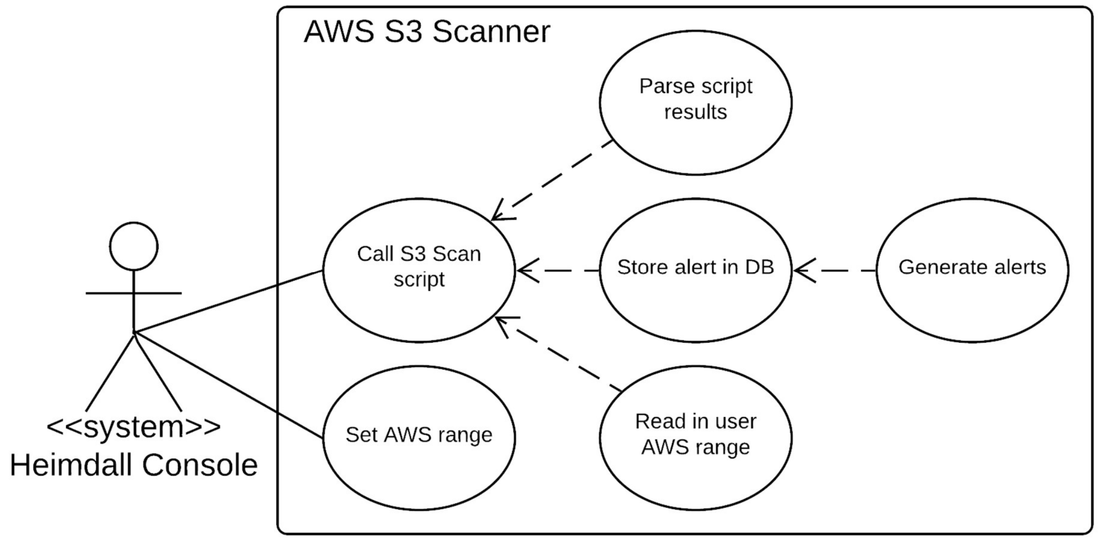

Figure 4 Use Case: AWS S3 Scanner

4.3.1 Set AWS Range

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The entered addresses are stored in the database

Main Success Scenario:

1. User is prompted with an address input box.
2. User fills out box and submits value.
3. System enters value to the database.

4.3.2 Call S3 Scan Script

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The script performs a scan on the complete range taken from the database. The results of the script are parsed to follow the standard format of the database, then if any results are of note an alert is generated for the user.

Main Success Scenario:

1. User send AWS scan request.
2. System retrieves the users associated AWS scan range.
3. System calls third part AWS script with the AWS range passed as an argument.
4. Parse the results of the script into a standardised storage format for the database.
5. Store the results in the database.
6. Prompt user with results.

Side steps:

2.1 User has no associated AWS addresses.

2.2 System prompts user with error and cancels script.

4.4 CVE Scanner

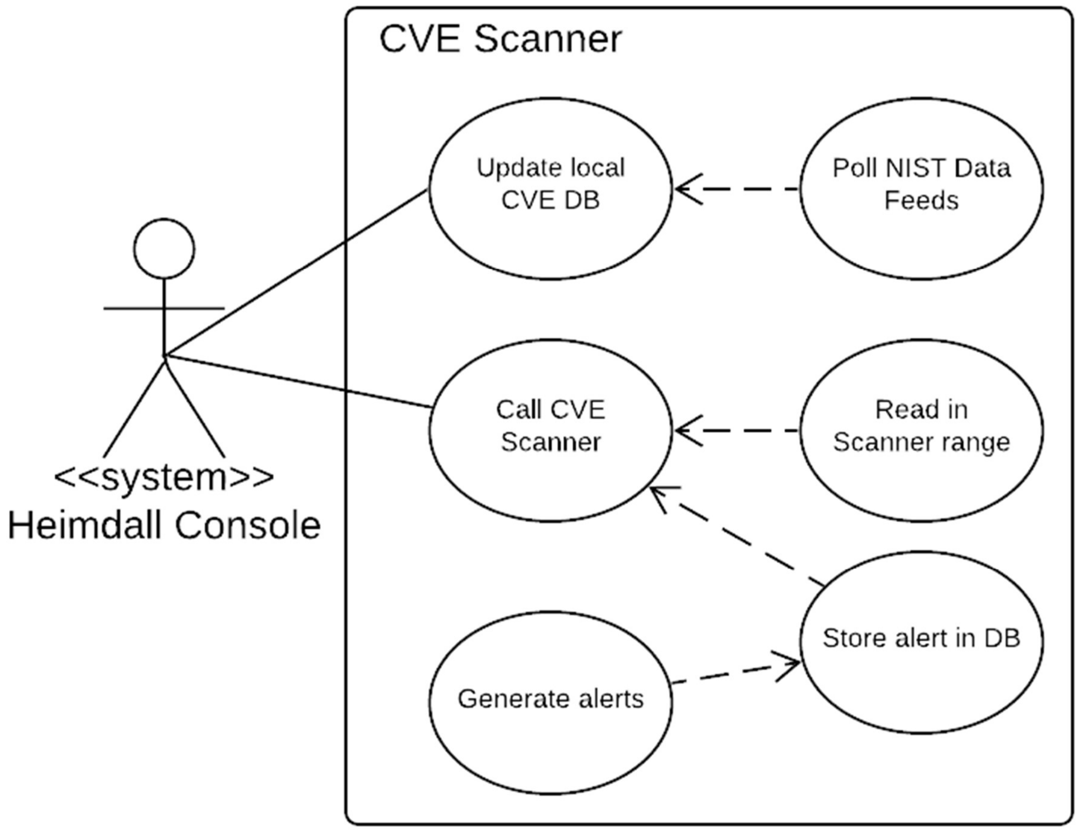

Figure 5 Use Case: CVE Scanner

4.4.1 Update Local CVE Database

Primary Actor:

Heimdall Console

Preconditions:

A standard period of time has elapsed, in this case the update will be performed every 3 days.

Success Guarantee:

The system will periodically poll NIST to ensure that the local version of the CVE database is up to date. When a scan is executed all IPs in the users given range will be successfully tested and the results stored in the database. If there are any results of note an alert will be generated for the user.

Main Success Scenario:

1. Issue API call to NIST website requesting current version information.
2. Poll local CVE database and compare version to API response.
3. Versions are different.
4. Issue API call to retrieve updates from ranging from the current local version to the current NIST version.
5. Enter the retrieved CVE data into the local CVE database.
6. Update local version information.
7. Log database update

Side steps:

3.1. Versions are the same.
3.2. Log that current versions is up to date.
3.3. Exit script.

4.4.2 Call CVE Scanner

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The script performs a scan on the complete range taken from the database. If the version of any assets correspond with vulnerable versions noted in the CVE database they will be flagged. The results of the script are stored in a local database, then if any results are of note an alert is generated for the user.

Main Success Scenario:

1. User sends CVE scan request.
2. System retrieves the predefined asset range for the requesting user.
3. System performs identification scan on the assets.
4. System logs any positive results in local database.
5. System alerts users to new vulnerabilities.
6. Exit script.

Side steps:

1.1 User has no associated CVE scan addresses.

1.2 System prompts user with error and cancels script.

4.5 Darknet Scanner

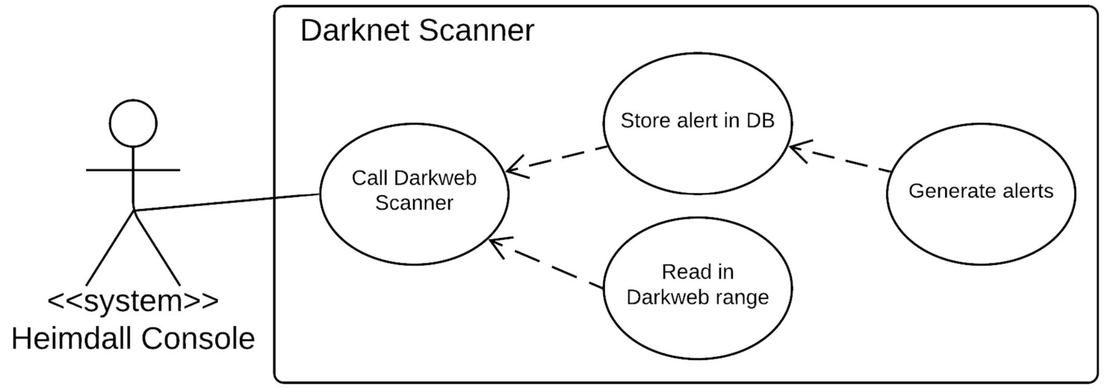

Figure 6 Use Case: Darkweb Scanner

4.5.1 Call Darkweb Scanner

Primary Actor:

Heimdall Console

Preconditions:

Actor initiating the use case is authenticated with the system and has permissions to generate reports.

Success Guarantee:

The script performs a scan on the complete range taken from the database. Any results are of note an alert is generated for the user.

Main Success Scenario:

7. User sends Darkweb scan request.
8. System retrieves the predefined Darkweb scan range.
9. System calls Darkweb script with the Darkweb range passed as an argument.
10. Store the results in the database.
11. Prompt user with results.

Side steps:

2.2. There is no Darkweb range defined.

1.3 System prompts user with error, cancels script, and alerts admin.

4.6 OSINT Scanner

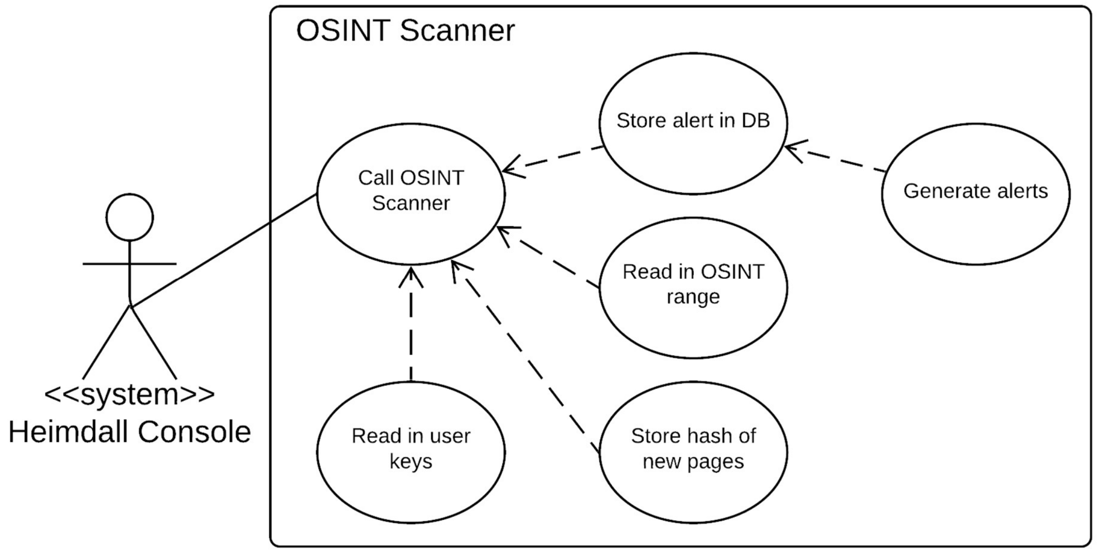

Figure 7 Use Case: OSINT Scanner

4.6.1 Call OSINT Scanner

Primary Actor:

Heimdall Console

Preconditions:

The Heimdall system will periodically invoke this script to ensure constant passive reconnaissance of sites, and alert on newly added keys.

Success Guarantee:

The script performs a scan on the complete range taken from the database. Any results are of note an alert is generated for the user.

Main Success Scenario:

1. OSINT Scan script will read in the User Keys from the database.
2. OSINT Scan script will read in the range to perform the scan from the database.
3. Script will request URL.
4. Scrape contents of page for any user keys.
5. Generate an alert if any user keys are in the content.
6. Store any links for the same domain found on the page if any.
7. Iterate though links as in steps 4 – 6 until all links are exhausted.
8. Script will repeat steps 3 – 7 until all items in the scan range have been scanned.

5 Site Map

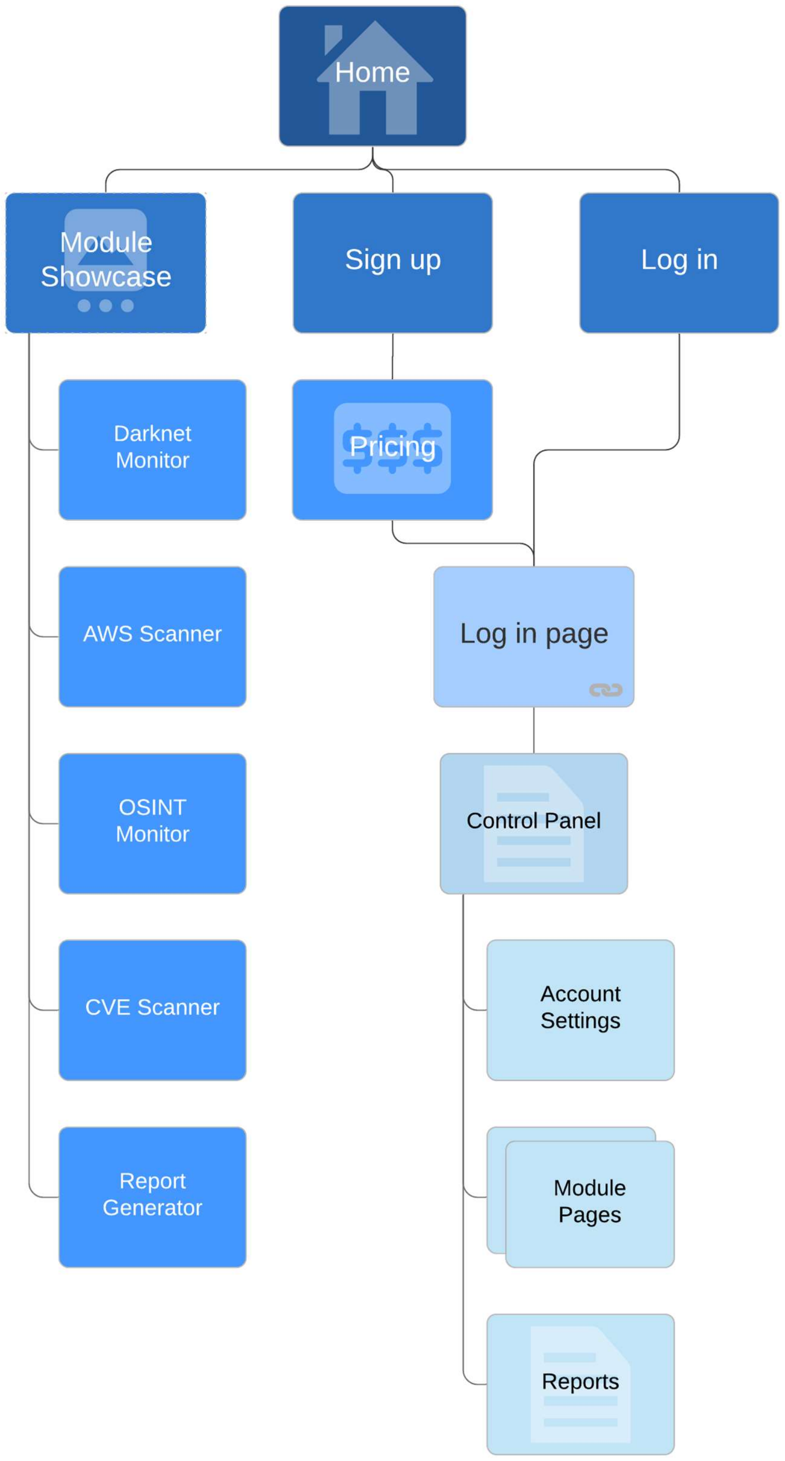
Home
Module Showcase
Sign up
Log in
Darknet Monitor
Pricing
AWS Scanner
Log in page
OSINT Monitor
Control Panel
CVE Scanner
Account Settings
Report Generator
Module Pages
Reports

Figure 8 Heimdall Site Map

6 FURPS+

FURPS+ is an acronym for the headings below. It is a model, designed by Robert Grady, to provide better context to the requirements of a project, both functional and non-functional. Non-functional in this instance means any criteria the system can be judged upon, not just specific behaviours.

6.1 Functionality

The functionality of the Heimdall Project has been laid out in sections 2 – 4 of this document.

6.2 Usability

The Heimdall web console will utilise a graphical user interface (GUI) to ensure that accessing functionality is as simple as clicking a button. The average user should have some base level of technical knowledge, however this is mainly for the initial configuration of the organisations keys, and the interpretation of alerts. This will all be performed by the use of clearly labelled input boxes to aid users. The API should be well documented to allow for third party developers to either integrate or develop new modules for the Heimdall platform.

6.3 Reliability

Heimdall is a web application and as such should maintain a high degree of uptime. It should be able to recover from failures in a timely manner with minimal impact. Heimdall addresses these issues by utilising a microservices architecture in place of a single monolithic application. In essence, Heimdall is composed of several standalone components that interact by passing data and invoking API calls. This means that if one component was to fail the system as a whole should be still able to function, although in a reduced functionality. The issue of recovery is addressed as Heimdall makes use of Docker containers to host the component services. These containers can easily be rebooted without any great effect on the system as a whole.

6.4 Performance

The web app should be responsive, able to fully load a page in under 1 second. The modules will take time to perform their functionality however, as such a processing icon should be used to show the currently running process.

6.5 Supportability

Heimdall should be able to perform on demand scanning and report generation. The instillation process will be made simple by the use of Docker containers, to run the application, an individual just has to download the source code from the remote repository, execute the Docker Compose script to allow the container to function together as a single application. The setup will also be configurable through the use of editable environment variables. As the project is made of multiple components interacting through API calls, it is very extensible. New modules could be developed utilising the internal API.

6.6 Additional

As Heimdall will function with several accounts, some form of authentication and session management will be implemented to allow for concurrent users. As Heimdall will make use of a RESTful API upon which third parties can develop or integrate new application, it should adhere to the OpenAPI standard to formalise the usage for developers. The requests and responses will transfer data in JSON format throughout the application, again this is to standardise usage and ease future development. If a module generates data in another form, such as XML or YAML, a parser must be implemented before the data is rent to the originator of the request.

7 Inspirations

7.1 Existing Applications

Fireeye Threat Intelligence is a threat intelligence provider who will monitor existing attack trends and provide contextualised intelligence to inform customers of potential threats. Their services are accessed through a web portal, inside of which clients can review intelligence and request analyst support.

7.2 Differences

Currently, to achieve some of the basic tasks of a security analyst investigating the above mentioned issues, an analyst must use a variety of independent tools, all disconnected from the investigation as a whole. This disunity can increase the work factor of a task, thereby reducing the effectiveness of an analyst. Analysts must also manually condense the results of these many independent tools into a report.

The unique point of Heimdall is its flexible threat intelligence and vulnerability identification ecosystem accessed through a web application. This system design allows for continuous development to both update existing modules, and develop new modules to target new sources of intelligence or utilise new techniques. This unification of tools into a single web accessible console and the automatic report generation will streamline work flows for analysts, potentially saving large amounts of time in the future.

7.3 Other Influences

The inspiration for the AWS S3 bucket scanning module came about as in recent years, there has been a rise in the accidental disclosure of company data through the misconfiguration of cloud storage [1]. Of these incidents the Amazon Web Service (AWS) has been the most common platform involved, specifically their S3 storage service, or “S3 buckets”. As more organisations transition to the cloud incidents of accidental disclosure will rise, to combat this a method of monitoring for exposure can be implemented to ensure a rapid identification and remediation in such an incident.

During my work experience in SunLife, a large multinational, I got experience working with multiple tools while performing investigations and threat intelligence activities. I became frustrated with the many disjointed tools and time I had to spend entering data for reports. This was where the initial idea for Heimdall and the report generation module emerged. The OSINT module was conceived as I came across multiple instances of proprietary information online, having been undetected by tools in use by the organisation.

8 Metrics

There are several key success metrics that are necessary for the system to function as detailed above. They are:

- Users can input ranges and keys necessary for scanning.
- Scans can be performed successfully and generate alerts.
- Reports are able to be generated from alerts on the database.
- The core Heimdall APIs allow microservice modules to interact with the platform.

The secondary success metrics, those which are not necessary for the project to be functional, are as follows:

- Users are able to register and sign into the application.
- Users are able to be assigned access to specific modules.
- Users are able to perform basic administrative actions on their account.
- Local CVE database can be successfully updated.
- User access to modules can be configured.
- Basic account administration.

9 Testing

An example report from an existing organisation should be attempted, whether this be IT Carlow, or SunLife. SunLife has been suggested as of the previous work experience I have undertaken there, and the connections I currently maintain with Digital Security and Threat Management in the organisation.

The OSINT Scanner could be tested against a site which is known to contain certain user keys. Permission has been gained to perform OSINT test scans on Mappa.ie.

The CVE scanner can be tested on a purposefully vulnerable system such as metasploitable.

Supplied user keys and ranges should be tested for accepting bad values.

References

[1] Rapid7, "There's a Hole in 1,951 Amazon S3 Buckets," 27 March 2013. [Online]. Available: https://blog.rapid7.com/2013/03/27/open-s3-buckets/. [Accessed 23 December 2019].