Attack Atlas

FINAL REPORT

Team 16
Client: Lotfi Ben Othmane
Advisor: Lotfi Ben Othmane
Team members:
Jacob Abkes
Dylan Black
Andy Dugan
Jack Phillips
Zhi Wang
sddec21-16@iastate.edu
http://sddec21-16.ece.iastate.edu/

Executive Summary

Development Standards & Practices Used

Our engineering standards are as follows:

- Split production from development
  - Prevents developer from accidentally deleting production data
  - Better test environment
  - Good control of the software version
- Developing in the same environment
  - Helps mitigate cross-environment confusion
  - Allows team members to collaborate more effectively
- Document before execution
  - We identified and documented our project use cases on a discovery basis, learning our requirements as we constructed the system
  - This prevents development from moving faster than user acceptance
- Iterative approach requires modifiability and modularity
  - Components have been designed in an object-oriented fashion, to maximize modularity/reusability and minimize coupling.

Summary of Requirements

Functional requirements:

- Ability for security experts to create blog posts
- Ability to search blog posts
- Ability to see a breakdown of different threat models based on the blog posts

- Ability to create incident reports (real life examples of cybersecurity threats) to be attached to relevant posts.

Environmental requirements:

- An authentication system which prevents unauthorized personnel from submitting blog posts, to ensure the sites hosts only accurate threat models
- Design a browser-friendly interface for users that can be accessed anywhere

Economic requirements:

- Uses university resources to circumvent costs associated with hosting and processing

- Switch away from wordpress to avoid third party plugins that could invoke costs

Applicable Courses from Iowa State University Curriculum

- SE 329 - Software Project Management
- SE 339 - Software Architecture & Design
- COMS 352 - Intro to Operating Systems
- SE 319 - Constructing User Interfaces
- ENGL 314 - Technical Communication
- COMS 311 - Introduction to Algorithms
- COMS 363 - Intro to Database Management Systems

New Skills/Knowledge acquired that was not taught in courses

- All knowledge relating to using and implementing Wordpress
- Most cybersecurity and threat modeling knowledge
- Knowledge of the React framework
- Knowledge of the Node js framework
- Workflow of a React + node js application
- Use of the React-Bootstrap framework
- Use of DraftJS for rich text support

Table of Contents

1 Introduction 5

1.1 Acknowledgement 5

1.2 Problem and Project Statement 5

1.3 Operational Environment 5

1.4 Requirements 5

1.5 Intended Users and Uses 7

1.6 Assumptions and Limitations 7

2 Design 7

2.1 Previous Work and Literature 7

2.2 Design Thinking 7

2.3 UI Design 8

2.4 Backend Design 8

2.5 Database Design 9

2.6 Project Architecture 10

2.7 Technology Considerations 11

2.8 Development Process 11

2.9 Final Product 12

2.10 Security Concerns and Considerations 15

3 Testing 16

3.1 Unit Testing 16

3.2 Interface Testing 16

3.3 Acceptance Testing 16

3.4 Results 16

4 Evolution 17

4.1 Evolution of Technology Requirements 17

4.2 Evolution of Functional Requirements 17

4.3 Evolution of Architecture 17

5 Closing Material 18

5.1 Conclusion 18

5.2 References 18

5.3 Appendices 19

Appendix I - Operations Manual 19

Appendix II - API Endpoints 20

Appendix III - Dependencies 26

List of Figures and Diagrams

Figure 1	8
Figure 2	9
Figure 3	10
Figure 4	11
Figure 5	12
Figure 6	13
Figure 7	13
Figure 8	14
Figure 9	15

1 Introduction

1.1 Acknowledgement

We would like to acknowledge Lotfi Ben Othmane for being not only the middle-man between our clients of cyber security experts, but also for being our adviser and giving us weekly advice and direction.

1.2 Problem and Project Statement

Problem

There is currently no centralized database that efficiently documents and organizes threat modelling patterns. A system such as this is necessary in the fight to ensure the safety and security of software systems around the world. The ability to easily gain the knowledge required to effectively mitigate current threats is something that many organizations and specialists wish they had, as such a database has the ability to adapt and change as new threats and vulnerabilities are discovered.

Solution

Our team has devised a web application that includes a database of documentation regarding various threat modelling patterns. This web application will allow users to submit knowledge regarding threat modelling patterns in the form of blogs, where these blogs will be efficiently organized so that a third party could more easily access the information they would need. Our application will also allow for users to interact with cybersecurity experts in the form of comments and voting on posts.

1.3 Operational Environment

Due to our project being software-based, any environmental hazards are limited to the server. The server is hosted at Iowa State during development, and a copy of all of the files needed to run the website are saved on GitLab. The website itself from a user perspective will be able to run on any modern web browser.

1.4 Functional Requirements

- When a user clicks ‘Register’ they will be able to input their desired username, email and password to create an account.
- If a user inputs a username that is taken, then the application will prompt the user for a different username.
- When a user has an existing account, they should be able to log into it.

- When a user opens the website they may view a blog post without logging in.
- When a user opens a blog post for a threat model, the application will show a post split into 8 sections, those being the username of the poster, the title of the post, the summary, context, problem, solution, alternate solutions and tags. The post will also have a section to view linked incident examples, and a comment section that shows the user comment, their username and the time which the comment was submitted.
- Where the application can support rich-text, the user will be able to link images, create bold and stylized text, numbered and bulleted lists. The following text fields support rich text formatting: Context, Problem, Solution, Alternate Solution, and Incident Examples.
- While a user is logged in, they have the ability to create incident reports (real life examples of cybersecurity threats) to be attached to relevant posts.
- When a user creates a post they will be prompted with 7 sections to fill in, the title of the post, the summary, context, problem, solution, alternate solutions and tags
- When a user clicks a tag within a post, they will be taken to the homepage in which the selected tag's content will be searched for them, displaying all posts with the same tag.
- If a user attempts to submit blank content to the application, it will be denied.
- When a user creates a post, they may add tags in a comma-separated list format.
- When a user creates a post they have the option to add as many incident examples as desired.
- While a user is logged in, the user has the ability to add comments to posts.
- When a user views a post, the post's view counter is incremented.
- When a user who is logged in rates a post as helpful or not helpful, their input is stored and the counter is incremented appropriately.
- If a user who is not logged in attempts to rate a post, they will be denied.
- While a user is logged in, they have the ability to add a single ✔ OR X to the counter for each post.
- If a user without the ability to create posts (users that are not verified cybersecurity experts) attempt to create posts, the system will prevent them from doing so.
- When a user searches for terms by content, all posts whose content contain one or more of the provided terms will be returned.
- When a user searches for terms by tags, all posts whose tags contain one or more of the provided terms will be returned.

- When a user clicks on the search by content/terms toggle, the toggle will shift to the setting not currently selected, and all further searches will reflect this change.

- While a user is logged in and viewing a post, they may create an incident example inline, and immediately associate it with the currently-viewed post.

- If a user is not logged in and attempts to create a new incident example through an existing post's page, the user will be denied.

1.5 Intended Users and Uses

We have two target groups of users. The first group are blog-posters, these users will want to create posts about threat models to inform others. This would also serve as a 'resume buff' for those individuals similar to stack overflow's impact scores and badges. The second group of users will be viewers. These users will not be interested in creating posts, although they will likely interact with posts and include their own insight. These users will oftentimes be active members of the industry. This second group of users will utilize the threat database to improve the security and planning of software projects.

1.6 Assumptions and Limitations

Assumptions:

- All users with login credentials will be verified cyber security experts
- Concurrent user count doesn't exceed 10,000.

Limitations:

- Application cannot function without users inputting threat modelling data

- Without a load balancer the application will suffer from bottlenecking.

2 Design

2.1 Previous Work and Literature

As Professor Lotfi Ben Othmane said himself, what we're doing in this project has no direct precedent. In regards to research, our group is not aware of any research that may or may not have led to the inception of this project.

2.2 Design Thinking

The definition process is focused on specifying a target to identify user needs and core issues, then reframing them to reflect new knowledge. Ideation is dedicated to diving into problems with teams to generate new ideas.

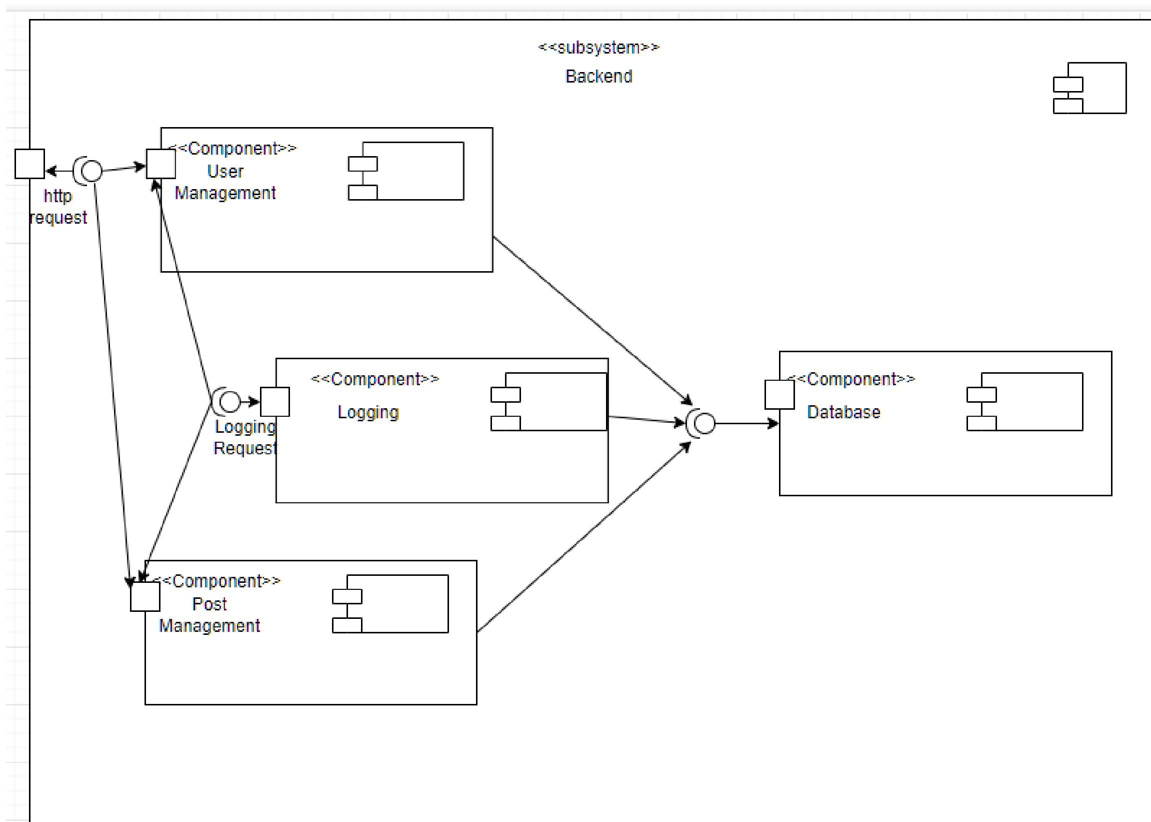
2.3 UI Design

The frontend is constructed using the React framework to allow for items ranging from individual page elements to entire web pages to be designed in an object-oriented fashion for maximum reusability. Rendering is done with raw HTML. Styling is primarily handled via Bootstrap, with custom CSS work done where necessary. Beyond this, the wysiwyg (what you see is what you get) library is used, built upon DraftJS, to provide users with rich text support. Dynamic content that relies on database values is asynchronously fetched through AJAX requests, made portable by jQuery. The react project gets compiled into 'static' html pages and javascript by web-pack, these files can then be served to a user on a request to our site.

2.4 Backend Design

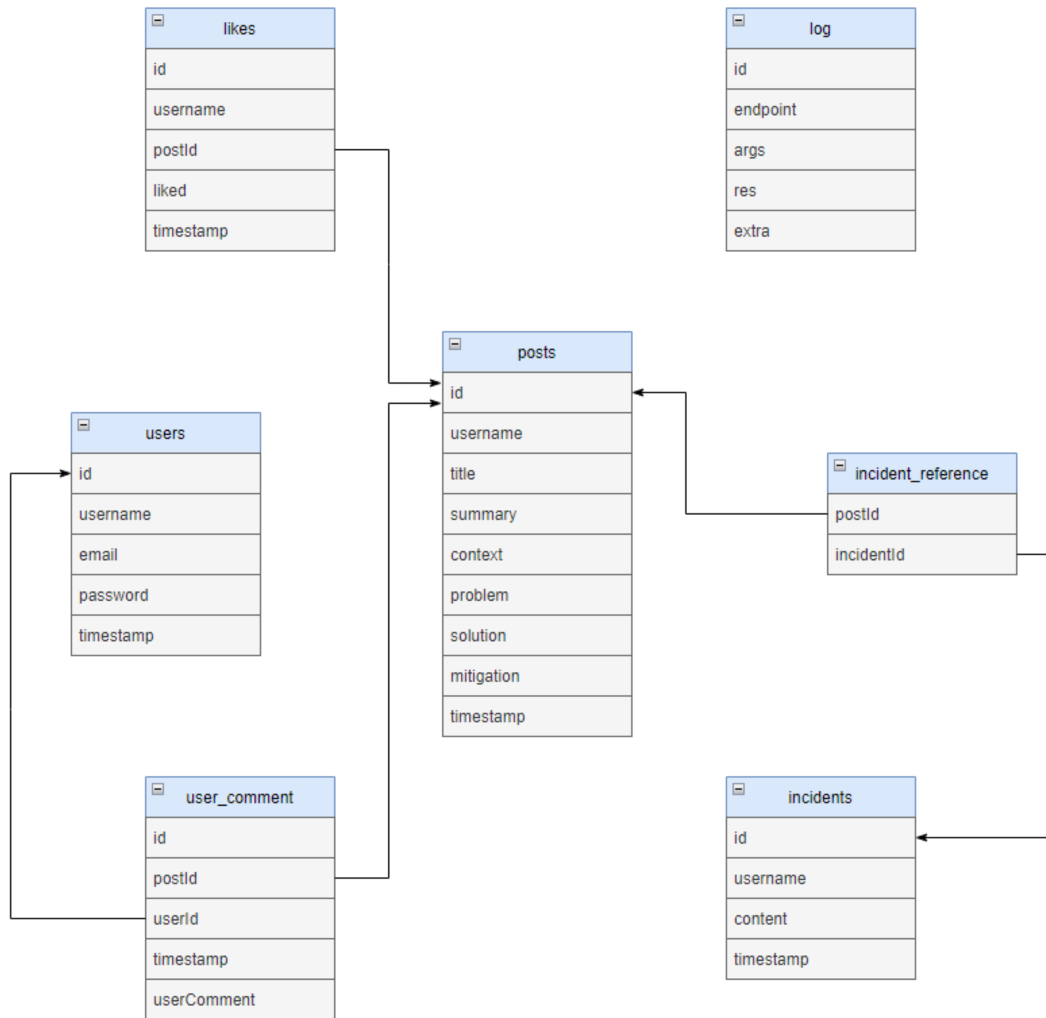
The backend is developed on a Node.JS utilizing an Express.js app. The backend consists of four main components. These components include a base entry component, user-management, post-management and an internal logging service. The backend was designed such that each component could be separated into its own service. With further development planned so that each service would be containerized and hosted in it's own instance. Figure 1 depicts the backend as a sub service when hosted on a single machine.

FIGURE 1



2.5 Database Design

FIGURE 2



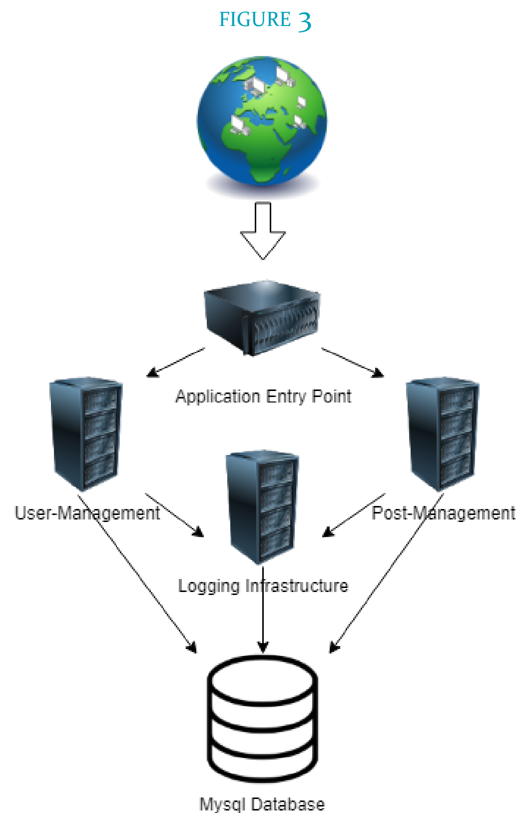
2.6 Project Architecture

The final project architecture is depicted in figure 6. It is separated into 6 parts, each part represents either an input or an output layer. The first layer is the initial input layer, the earth represents some user wanting access to the application. These requests come in the form of http requests.

The application entry point is responsible for serving the html files of our frontend. The application entry point also is responsible for delegating user http requests to the relevant services. This is done via reading the URI and having predetermined routes for each service. I.e. /posts/some/request goes to the post-management component, while /users/some/request goes to the user management service. This component was designed as a placeholder for a cloud hosted load balancer to improve the scalability and robustness of the application.

The User-management component is designed to manage user information and authentication. The Post-management component is designed to manage the workflow of a post. This would include storing the data, and disallowing certain actions for non-users on a post. The logging component is responsible for collecting http request information, and the success or failure of the request It would then store the information for auditing, maintenance, and debugging purposes. All information for our application is managed by a single mysql database to allow for easy interactions between components.

FIGURE 3



2.7 Technology Considerations

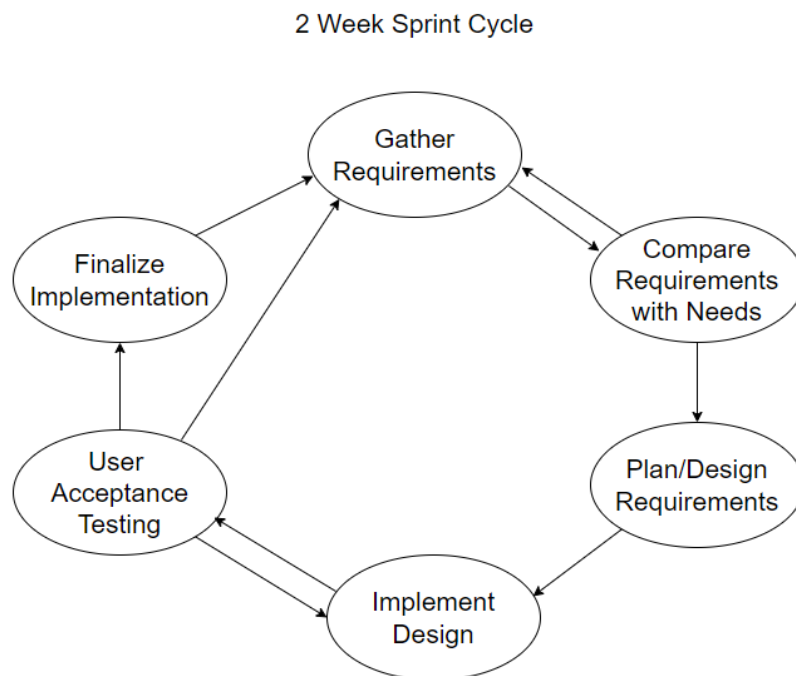
- The server for which the web application is running has limited computing resources, meaning a lightweight application is mandatory.
- Having a single entry point for the application would be a source of bottlenecking of the application. This entry point will have to be hosted on multiple servers one for each geographic location with significant traffic.

2.8 Development Process

The development flow of our application follows a two week sprint cycle as depicted in Figure 4. This process begins with discussing with our client the business requirements of the application(This is a weekly meeting where we also perform user acceptance testing of last week's implementations). The team then discusses the business requirements and translates them into product requirements, these requirements then become the basis of the design. Once the design has been completed we implement the design and perform user acceptance testing. Once the user has accepted the implementation and is satisfied, the implementation is added to the production copy of the application.

FIGURE 4

Our 2 week sprint cycle that we've followed for developing and integrating every feature for the web application



2.9 Final Product

Our website is split into multiple pages. In order to post, comment, or add incident examples, you must be logged in. Major navigation is all done through the navigation bar, where users can find the homepage (where users can search posts and view most recent posts on the sidebar), about page, login, registration, and finally the post submission page.

FIGURE 5

Threat model post submission page. Users must be logged in for a submission to go through, and all fields except the tags are required. Context, Problem, Solution, and Alternate Solution can be edited with rich text, allowing the use of common text editor features. The author can add existing incident examples by its unique number ID, or create a new one with an added rich text box.

FIGURE 6

Our finalized home page, with recent submission sidebar and a centralized list of all the threat models. Anyone can search either the content of posts or the tags of the post by using the search bar.

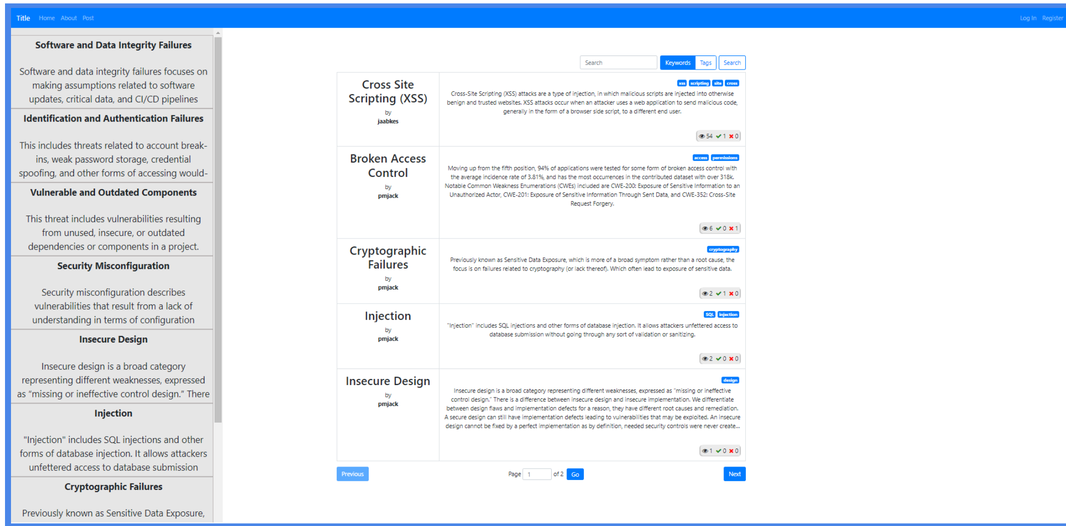


FIGURE 7

Example post page which displays content submitted through the submission page. From here, users can click on the tags to view other posts that share the same tags, see post view counts, as well as vote on how helpful the post is if you are logged in.

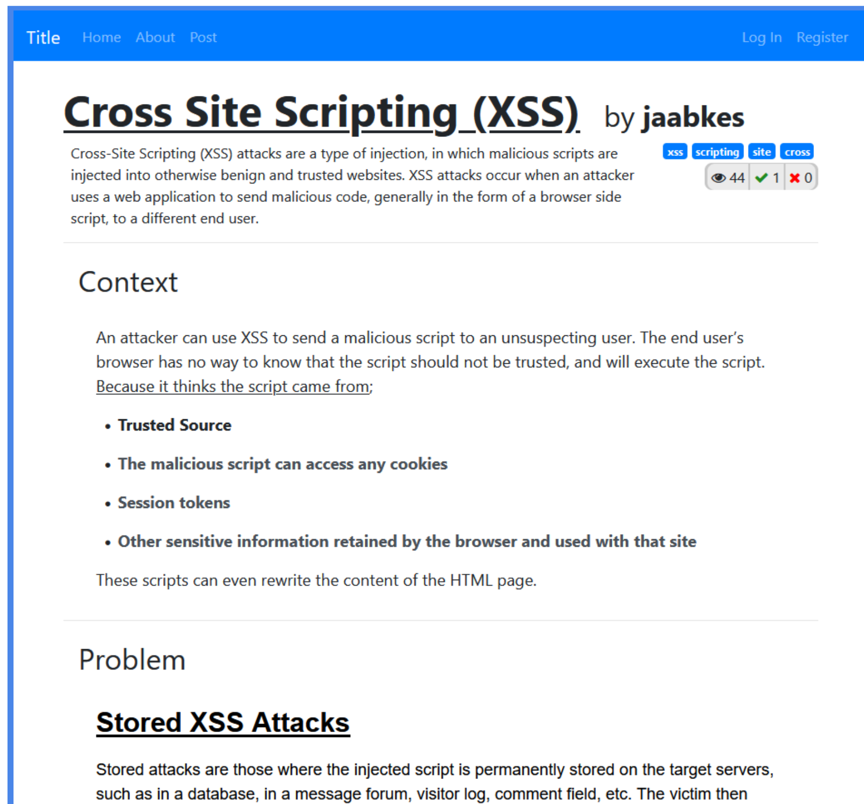


FIGURE 8

Bottom of the same example post page as Figure 7, showing incident example linking and comments. Logged in users can submit a new incident report (via a rich text box) or link an existing one using its unique number ID. Logged in users can also post comments on threat posts.

Title Home About Post Log In Register

Also, it's crucial that you turn off HTTP TRACE support on all web servers. An attacker can steal cookie data via Javascript even when document.cookie is disabled or not supported by the client. This attack is mounted when a user posts a malicious script to a forum so when another user clicks the link, an asynchronous HTTP Trace call is triggered which collects the user's cookie information from the server, and then sends it over to another malicious server that collects the cookie information so the attacker can mount a session hijack attack. This is easily mitigated by removing support for HTTP TRACE on all web servers.

The OWASP ESAPI project has produced a set of reusable security components in several languages, including validation and escaping routines to prevent parameter tampering and the injection of XSS attacks. In addition, the OWASP WebGoat Project training application has lessons on Cross-Site Scripting and data encoding.

Mitigation

Mitigations for XSS typically involve sanitizing data input (to make sure input does not contain any code), escaping all output (to make sure data is not presented as code), and re-structuring applications so code is loaded from well-defined endpoints

Incident Examples

Link an existing incident report by ID | Submit a new incident report. | ✖

Comment Section

Add a public comment

Submit

adugan 2021-11-16T22:51:57.000Z

Very well done

FIGURE 9

Login and register pages. In order to post, comment, or add incident examples, you must have an account and be logged in.

2.10 Security Concerns and Considerations

There were three main considerations taken into account for this project. The primary consideration was user-management. The team only implemented a prototype design of the user-management component as it is planned to leverage Google or Facebook SSO for account management. This reduces the overall risk of the project. Another consideration taken into account was user-authentication for access to the internal services. The current implementation has the ability to include oauth2 libraries for token management. The final consideration was the possibilities of sql injection. Although the team was able to find libraries for escaping characters in sql queries, alongside this there are several whitelisted inputs that only allow certain characters.

There was one main security concern for the website. This was the issue of Data Poisoning. Currently there is no solution to this problem, as we do not have a way of checking user inputted text for validity. This was also an issue as the web application's purpose was to act as a crowd sourced database for security vulnerabilities. In the future the application will need a way of keeping bogus data out.

3 Testing

3.1 Unit Testing

Due to the constantly changing requirements on our project, our team has held off on unit testing. The motivation behind this is that we will be changing the requirements and functionality of features. This means when requirements are changed we will be updating tests very frequently to maintain a fully-functional test suite. As mentioned, our development process has had weekly user acceptance testing which has helped us maintain our website.

3.2 Interface Testing

The main source of interface testing from our project was a shared collection of Postman requests. These requests and the API documentation maintained a single source of truth for interface development. We also utilized an internal service for logging. Logging statements are injected into the start of any backend request, and also into the success and failure callbacks. Our internal logging system recorded all information about the request including the specifics for successes and failures. See Appendix I for the full documentation of the API provided by our backend services.

3.3 Acceptance Testing

We communicated back-and-forth with the security experts collaborating with us on the project in order to achieve their desired functionality and look-and-feel. Our team will utilize the project advisor (Lotfi) as the middleman between us and the user acceptance testing.

3.4 Results

Each group member has installed a copy of the running environment to their local machines, so that most changes can be tested locally. For things that cannot be tested locally (such as database reads and writes), we coordinate running on our VM for brief periods of testing, ensuring that the stable branch goes live again once testing has concluded. In the future, we intend to use an additional server for beta testing, so that our live server can remain stable always.

The master branch of our application is the final 'production' copy of our software. This means any code that lives on this branch should have gone through thorough testing and review. This code could cause user issues, thus we must also have a dev branch for testing. This branch is less controlled and is used for testing your features on a production environment without actually polluting the production copy of our software.

4. Evolution

4.1 Evolution of Technology Requirements

We started the project with the intention of using Wordpress for basic blog posting. However, we very quickly found out that Wordpress does not provide the features we needed to fit our requirements. Namely, it didn't allow for things like post tags, comments, and predefined categories (Summary, context, problem, solution, alternate solution) without dealing with multiple plugins and configuring each to work with one another. In the end, we decided it would be best to manually create a blog posting system using React for our frontend and Node.js for our backend.

React and Node.js created a perfect base for our project. From there, we implemented a number of React libraries to handle different aspects of the website. To start, we used React-bootstrap to have a unified design language for the entire website. This allowed us to focus more on the functionality of the website while allowing React-bootstrap to automatically stylize most of the elements.

Other major libraries we used include Draft.js, a component developed by Facebook to allow for easy creation and storage of rich text, and react-draft-wysiwyg, a library built on top of Draft.js which implements a lot of the functionality of rich text (namely the toolbar and buttons that allow the user to select rich text elements like bold, underline, bullet points, etc.)

4.2 Evolution of Functional Requirements

From the outset of the project, our scope was much more ambitious: Our original intention for the website, similar to its current form, was a centralized location for cybersecurity experts to submit threat modeling data. Rather than being viewable by users, however, the data submitted would be parsed by a text mining algorithm and used to generate statistics and visualizations. This would never come to fruition, however, as the text mining algorithm, which was to be provided by an external source, never became available.

And with that, we changed course. Instead of statistical analysis, we instead shifted the design of our website to allow cybersecurity experts to submit formatted, human-readable information on threats directly. Now, our project takes the form of a platform for sharing and learning about modern cybersecurity threats and how one may mitigate or resolve them.

4.3 Evolution of Architecture

The initial architecture of our project was designed monolithically as a static website. This was due to the naivety of the team and lack of understanding of the initial requirements. Once the team had a better understanding of the project by the end of semester one, the architecture started moving to a less rigid structure. The team began developing the application to be an evolutionary architecture. This made the most sense as the client requested the team follow an agile workflow thus, the structure of the system was constantly changing.

The evolving architecture has taken shape to be a microservice architecture. This has allowed a flexible structure which delegates responsibilities to certain services. This also was shaped by the clients wanting to, in the future, move the hosting to a cloud solution particularly AWS. A

microservice allows for distributed cloud hosting to ease the scalability and robustness of the application.

The current build/deploy pipeline for the application consists of bash utilities created on the hosting virtual machine. The initial design of the project was wanting to use an automated CI/CD pipeline. The team chose not to pursue this as the return on time for setting CI/CD up was not worth it based on the timeline of the project.

5 Closing Material

5.1 Conclusion

So far we have created the first iteration of our web application which includes very basic forms of functionality regarding the implementation of our web server and database, along with a first implementation of WordPress. However, we decided to forgo WordPress due to it not fitting with our requirements as initially thought. As of writing the second version of this design document, we are basically restarting our project in terms of code. We are switching to the React framework in order to make a responsive front-end, and using Node.js as our back-end. This will involve redoing the front-end and back-end.

5.2 References

"Support – Official WordPress.com Customer Support." Support - WordPress.com. Accessed March 9, 2021. https://wordpress.com/support/.

"React – A JavaScript Library for Building User Interfaces." – A JavaScript library for building user interfaces. Accessed April 4, 2021. https://reactjs.org/.

"Jest · Delightful Javascript Testing." Accessed April 4, 2021. https://jestjs.io/.

"React-Bootstrap Documentation" Accessed April 22, 2021. https://react-bootstrap.github.io/.

"Overview | Draft.js" Accessed December 4, 2021. https://draftjs.org/docs/getting-started

5.3 Appendices

Appendix I

Operations Manual

Our project's source code can be found at https://git.ece.iastate.edu/sd/sddec21-16 (Note: This is only available to those with an Iowa State account.)

Frontend

- In order to run the front-end, you will need to have React and NPM installed on your system. From there, pull the project from Gitlab.
- Once the project is on your system, cd into the "website" directory and run "npm install" to install required dependencies. After that, cd into "website/client" and run "npm install" again.
- To start the website, within "website/client" run "npm start" to start running the front-end on your local machine.

Backend

1.) Software Required:
    a.) Git
    b.) MySQL
    c.) NGINX
    d.) NodeJS
    e.) PM2
2.) Configure NGINX configuration files to fit preferred server domain and ports.
3.) Clone git repository to desired directory.
4.) Run `npm install` from within the directory "{path to project}/website" to install all npm-managed dependencies not listed in step 1.
    a.) See Appendix II for the full list of direct dependencies.
5.) Build database via MySQL scripts supplied through the repository.
6.) To begin the server, run `pm2 start {path to project}/website/server/index.js`
    a.) By default, pm2 logs its stdout and stderr output to $HOME/.pm2/logs, in two separate folders.
    b.) In order to redirect the logs, use the following arguments when running the above command:
        i.) --log <filename>, redirects stdout and stderr to <filename>
        ii.) --output <filename>, redirects stdout to <filename>
        iii.) --error <filename>, redirects stderr to <filename>

Appendix II

API Endpoints

GET /posts/list

Description: Get all posts submitted

Request Body: Null

Response Body: [{

“submitId” : int,
“username” : string,
“title” : string,
“summary” : string,
“context” : string,
“problem” : string,
“solution” : string,
“mitigation” : string,
“tags” : string,
“timestamp” : DateTime
“likes”: int,
“dislikes”: int,
“views”: int
},...]

GET /posts/getMostRecentSubmission

Description: Get all posts submitted

Request Body: Null

Response Body: [{

“submitId” : int,
“username” : string,
“title” : string,
“summary” : string,
“context” : string,
“problem” : string,
“solution” : string,
“mitigation” : string,
“tags” : string,
“timestamp” : DateTime
“likes”: int,
“dislikes”: int,

“views”: int
},...]

POST /posts/submit

Description: Used for submitting threat model info

Request Body:{
“username” : string,
“title” : string,
“summary” : string,
“context” : string,
“problem” : string,
“solution” : string,
“mitigation” : string,
“tags” : string
}

Response Body: {“message”: “Submission Successful”}

POST /posts/comments/{submitId}

Description: Used for submitting comments to a post

Request Body:[{
“postId”: int,
“submitId” : int,
“username” : string,
“timestamp” : DateTime,
“userComment” : string
},...]

Response Body: {“message”: “Comment Uploaded”}

GET /posts/comments/{submitId}

Description: Used for getting comments of a post. Example below shows 2 comments for the 3rd threat model submission, while postId is unique to each post.

Request Body: None - but make sure the submitId is in the URI

Response Body: [{
"postId": 4,
"submitId": 3,
"username": "Phil",

"timestamp": null,
"userComment": "Good post!" },
{
"postId": 5,
"submitId": 3,
"username": "George",
"timestamp": null,
"userComment": "Needs more detail"
}]}

POST /posts/search/text:

Description: Used to get search results of threat model patterns

Request Body: {
“search” : “search term (user input)”
}

Response Body: [{
“submitId” : int,
“username” : string,
“title” : string,
“summary” : string,
“context” : string,
“problem” : string,
“solution” : string,
“mitigation” : string,
“tags” : string,
“timestamp” : DateTime
“likes”: int,
“dislikes”: int,
“views”: int
},...]

POST /posts/search/tag:

Description: Used to get search results of threat model patterns

Request Body: {
“search” : “search term (user input)”
}

Response Body: [{
“submitId” : int,
“username” : string,

“title” : string,
“summary” : string,
“context” : string,
“problem” : string,
“solution” : string,
“mitigation” : string,
“tags” : string,
“timestamp” : DateTime
“likes”: int,
“dislikes”: int,
“views”: int
},...]

POST /users/create:

Description: Used to create a new user (return null if error)

Request Body: {
“username” : string,
“email” : string,
“password” : string
}

Response Body: {
“username” : string,
“email” : string,
“password” : string
}

POST /users/login:

Description: Used to sign in to an account (return null if error)

Request Body: {
“email” : string,
“password” : string
}

Response Body: {
“email” : string,
“password” : string
}

POST /posts/action

Description: Invoked when a user likes or dislikes a post
Request Body: {
"username": string,
"id": int,
"liked": int
}
Response Body: {
Null
}

GET /posts/list/:id
Description: Get post with id :id -- adds a view to views when invoked
Request Body: None, but endpoint id var needs to be there
Response Body: {
"submitId" : int,
"username" : string,
"title" : string,
"summary" : string,
"context" : string,
"problem" : string,
"solution" : string,
"mitigation" : string,
"tags" : string,
"timestamp" : DateTime
"likes": int,
"dislikes": int,
"views": int
}

GET /posts/incidents/list
Description: Get a list of all incidents
Request Body: None
Response Body: [{
"id" : int,
"username" : string,
"content": string
"timestamp": DateTime
},...]

POST /posts/incidents

Description: Submit a new incident

Request Body: {
"username" : string,
"content": string
}

Response Body: {
"insertId": int
}

GET /posts/incidents/:postId

Description: Get incidents related to a post

Request Body: :postId

Response Body: [{
"id" : int,
"username" : string,
"content": string
"timestamp": DateTime
},...]

GET /posts/incidents/incident/:incidentId

Description: Get individual incident

Request Body: :incidentId

Response Body: {
"id" : int,
"username" : string,
"content": string
"timestamp": DateTime
}

POST /posts/action/associate/:postId

Description: Associate incoming list of incident ids with post id

Request Body: {
"incidentIds": array
}

Response Body: "Success"

Appendix III

Frontend dependencies:

- Bootstrap
- DraftJS
- FontAwesome
- jQuery
- React
- web-vitals
- React-draft-wysiwyg

Backend dependencies:

- body-parser
- Express
- loglevel
- Moment
- Nginx
- NodeJS
- MySQL