**Senior Design Final Report**

**Home and Office Security Scanner (HOSS)**

Icon

Description automatically generated

Version 1.0 - 05/12/2023

Team Members:

Ricardo Castellanos

Luis Cortez

Diego Flores

Adrian Holloway

Jivan Karapetian

Minsu Lee

Asahel Monroy

Ashley Pablo – Ramirez

Garrett Stewart

Guang Wu

Faculty Advisor:

Dr. Huiping Guo

Liaisons:

Dr. Anson Fong

Sponsored By: Los Angeles International Airport

**Table of Contents**

1. Introduction 2
   1. Background 2
   2. Design Principles 3
   3. Design Benefits 4
   4. Achievements 5
2. Related Technologies 6
   1. Existing Solutions 6
   2. Reused Products 6
3. System Architecture 7
   1. Overview 9
   2. Data Flow 9
   3. Implementation 10
4. Conclusions 14
   1. Results 14
   2. Future 14
5. References 16

**1. Introduction:**

**1.1. Background:**

Merriam Webster defines Internet of Things as “the networking capability that allows information to be sent to and received from objects and devices (such as fixtures and kitchen appliances) using the Internet”. We live in an era where at any given moment, we are surrounded by the Internet of Things (which we will denote as IoT). In our household or office, certain IoT devices we have are our laptop, Smart Television, video game console, modern vehicle, router, Amazon Echo, and far more. All of these devices have some sort of network connectivity, whether via Bluetooth, Wi-Fi, or LAN. Due to this connectivity, silent opportunities arise for malicious activity in the form of hacking, network penetration, or unauthorized remote control. Due to the plethora of devices in one’s home/office at any given moment, it’s impossible for the average person to keep track of the privacy and security of each device. Which of your devices have the capability of spying on you, whether through audio or video? Which of your devices has a vendor who has experienced a cyber-attack in the past? These are questions that every IoT device owner is entitled to know and is where the Home and Office Security Scanner comes in, which shall be denoted as HOSS throughout the entirety of this report.

HOSS is a software-based network scanner which shall scan all IoT devices within a user’s network. HOSS then returns a list of all detected devices, along with an assessed number-based score for both a device’s security and privacy. The unique aspect of HOSS, which shall be discussed further when comparing it with other applications, is how it assesses a device in terms of both how secure it is from malicious attacks, and how private the device is in terms of said device’s brand selling the user’s data or not. The ever-expanding list of IoT devices was our motivation to tackle and solve the problem of combating the increased likelihood and cyber attacks against these IoT devices. Below is an overview of each of our group member’s contributions to HOSS.

* Asahel Monroy contributed extensively to our Software Requirements Document, our Software Design Document, as much of the HOSS’s client-side functionality and our device database (which shall be discussed in our system architecture section).
* Garrett Stewart heavily contributed to the server-side and back-end of our project, managing both our database transactional functions of HOSS, along with overall framework design, implementation, and debugging.
* Diego Flores contributed with our front end team, developing both a functional navigation bar and device glossary (which shall be discussed in our system architecture section). Diego also assisted with the Software Design Document and Software Requirements Document, along with device research for our database.
* Adrian Holloway is the team leader of HOSS. He coordinates all team meetings and communication both amongst peers and with our project advisor and liaison. He also assisted in the writing of this report, along with our Software Design Document and Software Requirements Document, along with back-end research.
* Ricardo Castellanos contributed to the user-interface design of HOSS’s front-end. He provided the initial design of our software, along with its current implementation in terms of virtually all client-side pages and logical flow. He also assisted with the Software Requirements document and Software design document.
* Luis Cortez heavily contributed to the architecture of HOSS in its entirety. He is responsible for the implementation and functionality of our network scanner and its communication with both our server-side and client-side. He also greatly contributed to the functionality, programming, and debugging of our client-side framework and server-side framework.
* Ashley Pablo-Ramirez contributed to device research and our Software Design Document. She also assisted with both designing the Home Page of HOSS, along with designing our project poster.
* Minsu Lee assisted with our Software Design Document and Software Requirements Document. He also contributed to front-end and client-side functionality, such as the About Page. Along with Ashley, he also assisted in designing the project poster.

* Guang Wu assisted in all areas of documentation, such as the Software Requirements Document and Software Design Document. He also assisted in server-side responsibilities such as debugging, framework design, and device research.
* Jivan Karapetian is the secretary of HOSS. Jivan is responsible for documentation such as the Software Requirements Document, Software Design Document, and the writing of this Project Report. In terms of secretarial activities, Jivan is responsible for the Meeting Minutes for our Friday meetings with our liaison and advisor, along with back-end device research.
* All group members contributed to our PowerPoint presentation slides in equal capacity.

**1.2. Design Principles:**

Our Framework indicating the flow of data and its relative modules shall be our primary deliverable and frame of reference for our design. It is the structural backbone of HOSS and will provide a visualization of how HOSS communicates with various application programing interfaces (while will be denoted as API) to achieve its desired output. We have also ensured that our code, data flow diagram, and overall framework is both easy to follow and can act as a boilerplate code for future projects. HOSS is both programmatically and practically straightforward in both server-side and client-side functionality. While the implemented code has consisted of much revamping, debugging, and research, the flow of logic through several classes and APIs is very intuitive and devoid of spaghetti code. Since HOSS requires access to a user’s network in order to scan their relative devices, it is unable to function solely as a web application which you can access as a website. Thus, HOSS will operate as a software which the user will run as a web application, ensuring efficient scanning and communication with the user’s router, with minimal network and router configuration from the user. Another reason why HOSS needs to be intuitive boiler-plate code is because it communicates with two APIs which shall be discussed in our Framework and Data Flow Diagram, which are Fing and the National Vulnerability Database. Any potential updates to those APIs means that maintenance and updating of HOSS’s code cannot be complicated.

**1.3. Design Benefits:**

Since one of the main goals of HOSS is to provide an intuitive framework both programmatically and visually, its ease of use and updating is a benefit of its design. Over the course of HOSS’s development, we have gone through several device scanning APIs, such as Network Mapper (NMAP). While NMAP is useful for many different problems that entail device detection, we noticed it didn’t exactly suit our needs by itself since we needed an API which returned as much information about a device as possible, such as Fing. During the addition of Fing to NMAP, the large majority of our code remained unchanged due to us previously foreseeing the potential need to modify this aspect of our framework.

Our database which we’ve incorporated into our software, MongoDB, also heavily complements the versatility and simplicity of HOSS. MongoDB is the database which stores all information regarding the user’s devices and their respective attributes. MongoDB saves device information in JavaScript Object Notation, which is known for being very human-readable. MongoDB is also very fast and can store big data, which is inevitable for HOSS due to the large amount of IoT devices it stores and will need to store in the future. Many devices can provide MongoDB information in many different shapes and sizes, which is what MongoDB is known for. Our database can store device information in varying fields, content, and sizes. This is in contrast to other query-based languages such as SQL, which requires context of fixed size and fields. The exact implementation of MongoDB in our framework shall be further discussed in the architecture section.

The client-side of HOSS is also very readable, user-friendly, and intuitive to users of all backgrounds, regardless of technical experience. Since the server-side of HOSS does the entirety of the technical and programmatic requirements, the user-side simply necessitates a simple button-press to scan the user’s network. After this scan, HOSS presents the security and privacy score to the user, and a glossary which explains all the technical terminology pertaining to the scan in an easy-to-understand manner. The server-side and client-side are both very scalable and versatile, which allows for further improvements by both the HOSS creators and anyone who would like to use our code as a boiler-plate for future projects.

**1.4. Achievements:**

Throughout Fall 2022 and Spring 2023, we have been able to generate a satisfactory evaluation of any user’s devices in terms of both security and privacy. In the very beginning of Fall 2022, we weren’t sure how we’d be able to accomplish gathering so much information about a device’s model name, connectivity method, and media access control (MAC) address. But as of May 2023, those are all the pieces of information we are able to gather from a large majority of IoT devices. Moreover, an initial concern of ours was how we would assess a device brand’s history of cyberattacks without manually researching each and every device. This concern was also remedied with us achieving automated research of a device brand’s model history through an API which directly links HOSS’s framework with the Common Vulnerabilities and Exposures (CVE) database, allowing HOSS to cross-reference a brand with its historical number of security vulnerabilities. In our Mongo database, we have also stored an extensive amount of device information from preliminary scans done by our group members, which will allow us to quickly and efficiently cross-reference these stored devices with any other user’s respective devices detected from our scanner. This will allow for a constantly faster and more detailed scan, the more HOSS becomes utilized. These are all the achievements for our server-side.

In terms of our client-side, we have a fully running application available for download and use. Our HOSS application comes with a Scan page, About page, Devices page (which contains all past devices scanned in a User’s network), and Glossary page containing definitions on cybersecurity terminology the average user may not be familiar with. The user interface and architecture for our HOSS application has been kept as user-friendly as possible to allow for further expansions and adjustments in the future. All User device information relating to MAC address, connectivity method, model name, and cyberattack history has been decoupled from the client-side and is all handled by the server-side in its entirety, allowing for a more streamlined user experience. All device information is available thanks to both our Fing API and CVE API, meaning that future updates to our APIs, which we have done in the past, may lead to much more detailed device information and evaluations in the future. However, our team has achieved a great amount on a technical level on both server-side and client-side operations.

**2. Related Technologies:**

**2.1. Existing Solutions:**

The concept of a device scanner in and of itself is not a new solution and has been around for many years. However, we have looked into some existing scanning applications which identify vulnerabilities in devices and assess what HOSS does differently. Two applications are Trend’s Housecall and Bitdefender.

* Both Bitdefender and Housecall are fundamentally very similar, in that both scan a user’s devices in their network and output a list of vulnerabilities each device has, along with solutions on fixing these vulnerabilities. Both applications assess the severity of a vulnerability ranging from low-risk to high-risk, and both base security vulnerabilities on the type of cyberattack which can be performed on a vulnerabile device.
* HOSS has a different approach to assessing the vulnerability of its devices. It provides a score of 1 to 5 in terms of how secure and private a device is. This leads us into the next difference HOSS provides, which is the privacy score. HOSS assesses if a device’s brand sells your data and bases the privacy score around this concept, as opposed to simply focusing on how secure a device is from cyberattacks and remote control. We also believe that HOSS can perform its scan faster than other network scanning applications due to its optimized API communication and simple framework structure. Housecall and Bitdefender are also simultaneously anti-malware software which requires more work from a user’s system to perform a full scan, whereas HOSS is solely a device scanner.
* Bitdefender and Housecall are fantastic device scanning applications, however we were not able to figure out what API’s they used, if any. We also agreed in the planning stages of HOSS to set ourselves apart by assessing a device’s privacy as well and providing a tangible, numbered score in order to provide an extra layer of understanding to the user. Due to our insistence with providing a privacy score, we were unable to locate an existing solution that performs this assessment as well. As of the writing of this report, we believe that is the one aspect which makes HOSS unique, in that a cross-reference a device’s model with its brand to determine if a user’s personal information is safe from being shared or sold with other vendors or companies.

**2.2. Reused Products:**

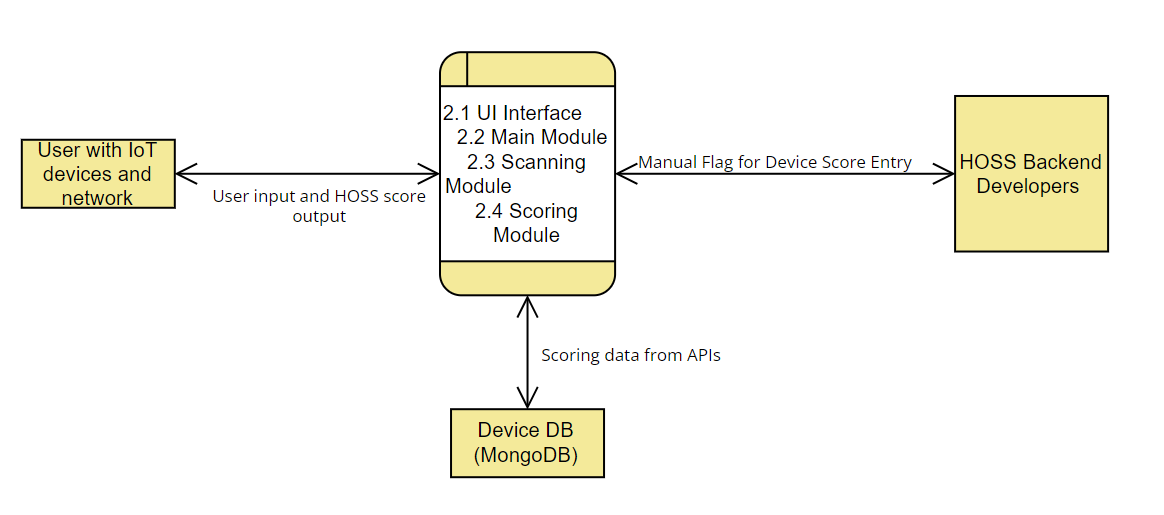
HOSS is developed in Python for all server-side operations and HTML/Cascading Style Sheets (CSS) for all client-side operations. These choices were made as a matter of preference from our team, and most, if not all, server-side operations can be achieved through various other high-level programming languages. The front-end/client-side operations are most easily achieved through HTML and CSS since it is the standard web-development language we have learned in our Web Programming courses at Cal State La. However, MongoDB was chosen as our database query language due to its versatility with allowing for data to be stored in JSON documents which is independent of a query’s length.

**3. System architecture:**

**3.1. Overview:**

The architecture for HOSS and its external users can be broken down into three main factors: the User (who will interact with HOSS and owns IoT devices), the Framework (all necessary modules required to complete), and the HOSS backend team (our server-side HOSS developers who will manually score devices that we do not have any vulnerability information on).

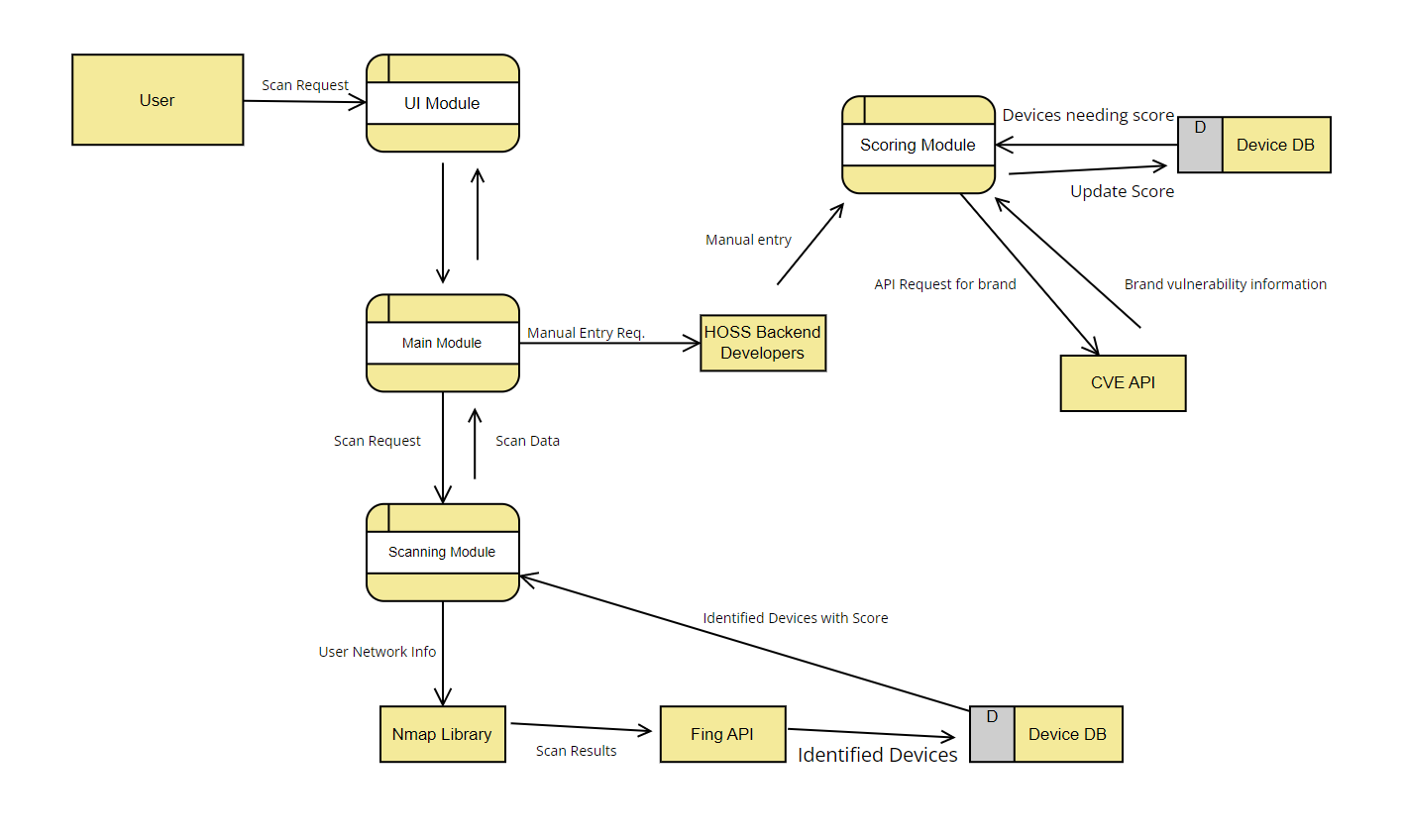
Below is our DFD Level 0, which is a consolidated diagram of our application at a high level



* **User with Iot Devices and Network:** Our first external agent is the User (who will request a scan through HOSS). We require that our User be connected to a home network, whether on ethernet or Wi-Fi. Also, our User must own at least one IoT device in order for our scan to detect and evaluate a device for the User. The User’s sole interaction with HOSS is by requesting a scan by pressing our “Scan” button. After this, the User’s interaction is complete, to which HOSS will return the User a list of their devices and their respective Security and Privacy scores for each device.
* **The Framework (The User Interface and all Applicable Modules):** This is the structure of HOSS and its respective operations. All modules will be discussed further in the Data Flow Section, but to summarize, we have: The User Interface Module, Main Module, Scanning Module, and Scoring Module. All modules are pertinent to providing the correct output for our User. Interacting with our Framework is the Device DB, which holds all historical information of both devices detected in our scans, and the respective security and privacy scores of these devices.
* **HOSS Backend Developers:** The final external agent interacting with HOSS is our developers for server-side operations. If an IoT device requires a security score our Device DB does not currently have, a flag is sent to our server-side developers in order to utilize our CVE API to return a list of all known vulnerabilities for a device’s brand. Our scoring module then calculates a weighted average score based on the total of all known vulnerabilities, to which our back-end developers then input this average into our Scoring Module to compile a Security Score, which is then stored in our Device DB for future use by the scanning module.

**3.2. Data Flow:**

Here is an overview of HOSS as a system in the form of a DFD-Level 1, showing all points of data flow throughout the entire framework of HOSS.



There are 4 major modules in this system. Below is an overview of each module and the respective external entities relevant to these modules.

**User Interface (UI) Module:** The User Interface Module is essentially the module and all its related Javascript classes pertaining to HOSS’s UI that our User will interact with. It is the simplest module which contains mostly static web pages with minimal involvement to the server-side. Our UI shall contain a Scan page, About page, Contact Us page, and Glossary page. These pages are all inherent to the UI module. However, the only page which shall allow for any User input and subsequently, data flow to the server-side, is the Scan page. Once the User clicks the Scan button, a Scan Request is sent from the UI module to the Main module.

**Main Module:** The Main Module is essentially the middle-man of all other modules and external entities. It is the first point of contact for obtaining the scan request from the User. Once the main module obtains the Scan Request, it transports this data to the Scanning Module (which shall be discussed next). The main modules two other responsibilities are to receive parsed scan data from the Scanning Module, and to send a flag to our HOSS server-side team once a device within in the parsed Scan Data is detected to not have sufficient vulnerability information, which consequently means there is no Security or Privacy score associated with it.

**Scanning Module:** The Scanning Module is the first non-middle-man module which performs the appropriate parsing of the scan request. The Scanning Module sends the user network information (which is the IP address of the user’s network to which all surrounding IoT devices are connected to) to our NMAP library. The NMAP library initially returns a list of all devices within the user’s network. However, since NMAP does not have as consistent of information as we had initially hoped regarding device model and connectivity details, we then require NMAP to send its parsed scan results to our Fing API. Fing fills in the blanks for all detected devices that do not have a model name or connectivity type associated with them, in order to provide as complete information as possible regarding which devices are connected to a User’s network. For example, if NMAP would simply tell the user that they have an Apple device connected to their network, then NMAP combined with Fing would tell the user they have an Apple Macbook Pro connected to their network.

Fing then sends the complete list of identified devices to our Mongo Device Database (MongoDB). MongoDB cross-references the list of devices to all of our previously stored devices to check if we have a Security and Privacy score associated with each device. It then returns a list of these devices and their respective Security and Privacy scores. It also identifies devices that we do not have information on regarding vulnerability history, in order for our back-end developers to know which devices manual research needs to be performed on.

**Scoring Module:** The Scoring Module essentially calculates a score from 1 to 5 on how secure and, separately, how private a device is. The security score is determined by a few criteria. These criteria include: Encryption Method, Communication Protocols, and Default Settings. For the Encryption Method, each known encryption method is assigned a score from 1 to 5 based on how secure this encryption method is. The same goes for communication protocols, where each communication protocol is assigned a score from 1 to 10 based on how effective the communication protocol a device uses is from outside attackers. For Default Settings, we look for 3 factors: If a device has automatic updates set by default, if a device has 2-factor authentication set by default, and if a device has a unique device identifier. The presence of each and any of these factors would improve the security score. The final criteria is how many vulnerabilities have been discovered by a device’s brand in the past. With direct communication through our CVE API, HOSS will automatically return the count of how many vulnerabilities a device’s brand has had in the past. The number of these vulnerabilities will then be weighted ,and an average number will be provided and factored into the other aforementioned scoring criteria. After all the aggregation and calculations are complete, the security and privacy scores are stored in the Device DB to both be outputted to the User, and for easy access for future scans of the same device and model.

Our Privacy Score is determined, firstly, by if a brand sells the User’s data or not. This is also achieved through manual research where our server-side developers will have to manually research each brand’s privacy policies. Additionally, privacy score shall be determined by what type of sensitive information a device contains about the User. These pieces of information include, but are not limited to: Social Security Number, payment information, date of birth, and address. These are also factors are server-side must manually research and input for each device since we are unable to automate this aspect at this time.

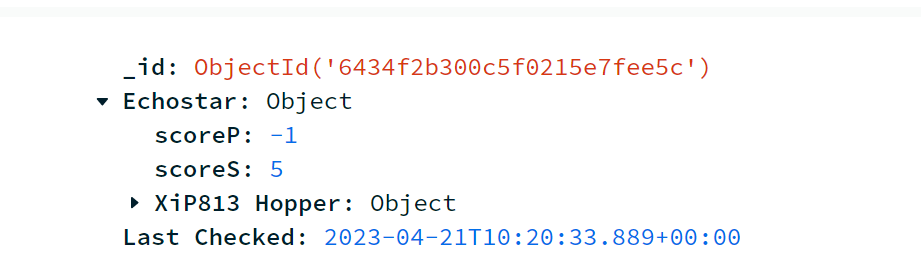
**3.3. Implementation:**

**3.3.1. Scan Data**

Scan data is collected from the user using NMAP. The NMAP library performs an Address Resolution Protocol (ARP) scan which pings all the devices in the user's network and gathers crucial information about the respective device in order to properly identify them. With this gathered information, we format it and send it in an API request to Fing's Device Recognition API to identify the brand, model and type for the device. Once we get the response, we query our MongoDB and pull the respective scores for each device.

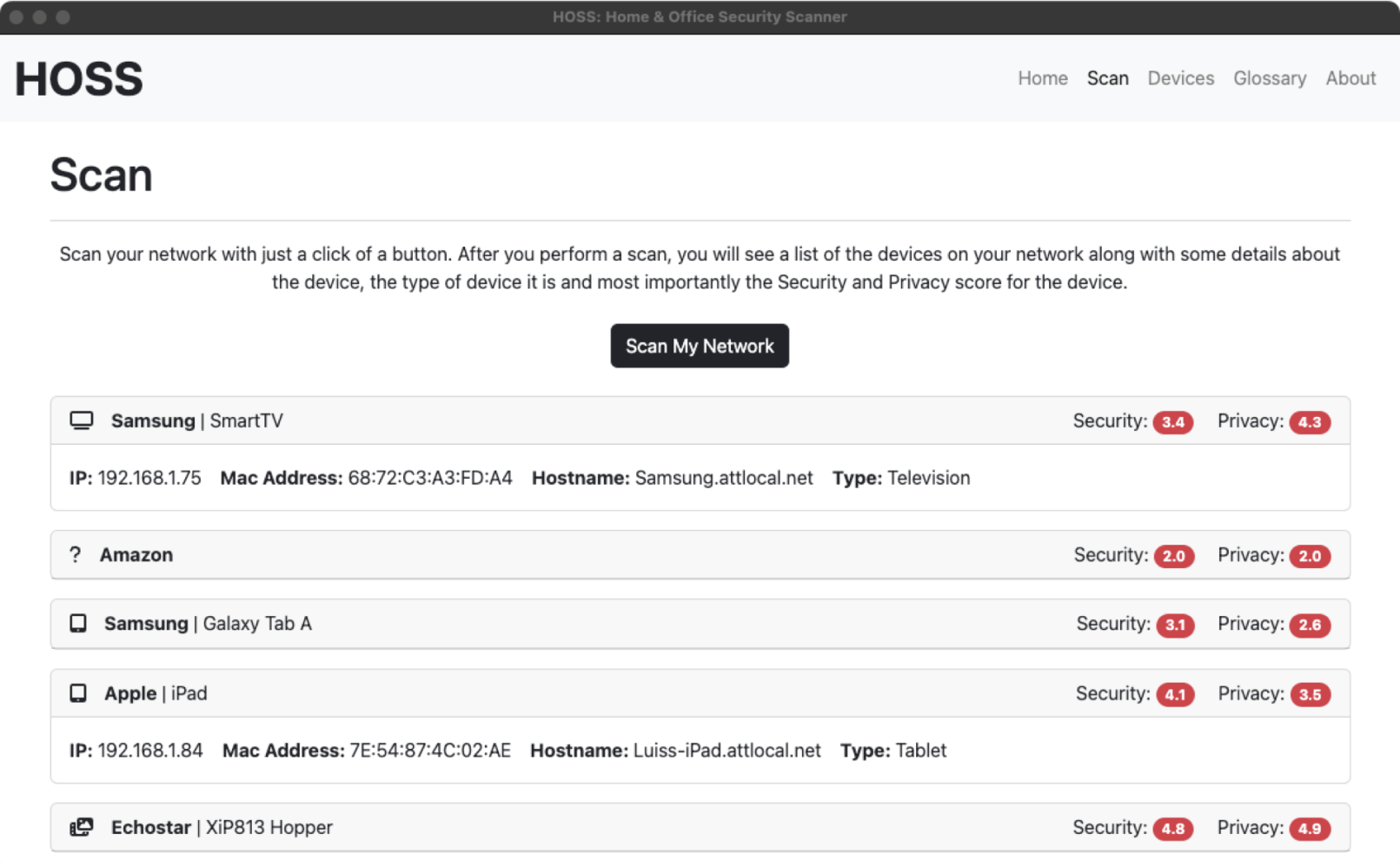
**3.3.2. Local Data Management**

To optimize the scanning operation, after the user performs their first scan, a JSON file will be stored on the user's device. This file will have the data from the original scan and when another scan is performed, the scanning module will cross reference this file and use this data, instead of making unnecessary requests to Fing's API. An example of a device stored in our MongoDB collection is shown below. In this example, our scan detected an Echostar brand device. HOSS then determined it was the model type “XiP813 Hopper”. Since we have already conducted manual research regarding the security of this device, the security score has already been assigned a ‘5’ by default. However, since we do not have information pertaining to Echostar selling our User’s information, the Privacy Score is assigned a ‘-1’ by default. The ‘-1’ serves as the flag to our server-side developers that manual research needs to be conducted and inputted regarding Echostar’s privacy.

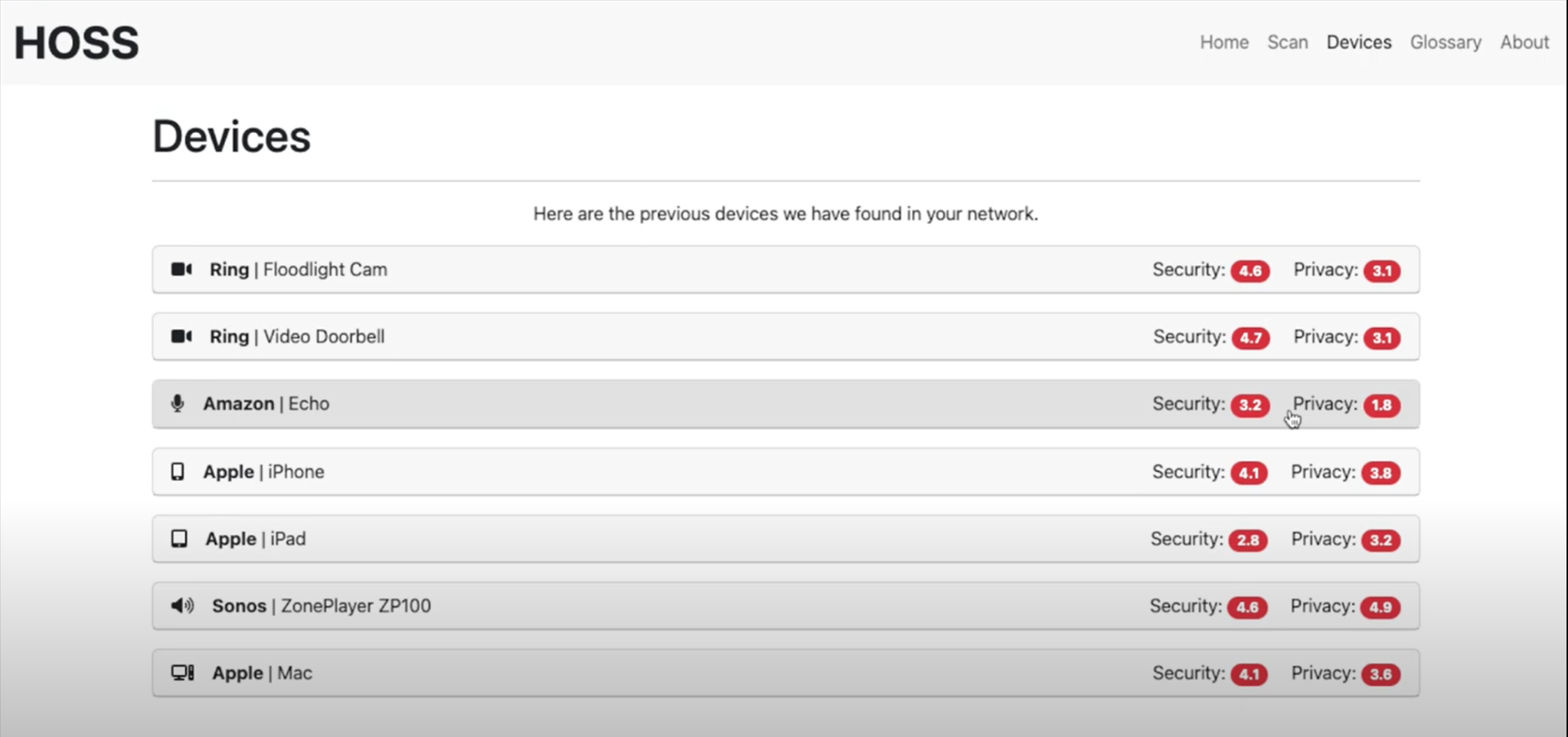


**3.3.3. User Interface/User Experience**

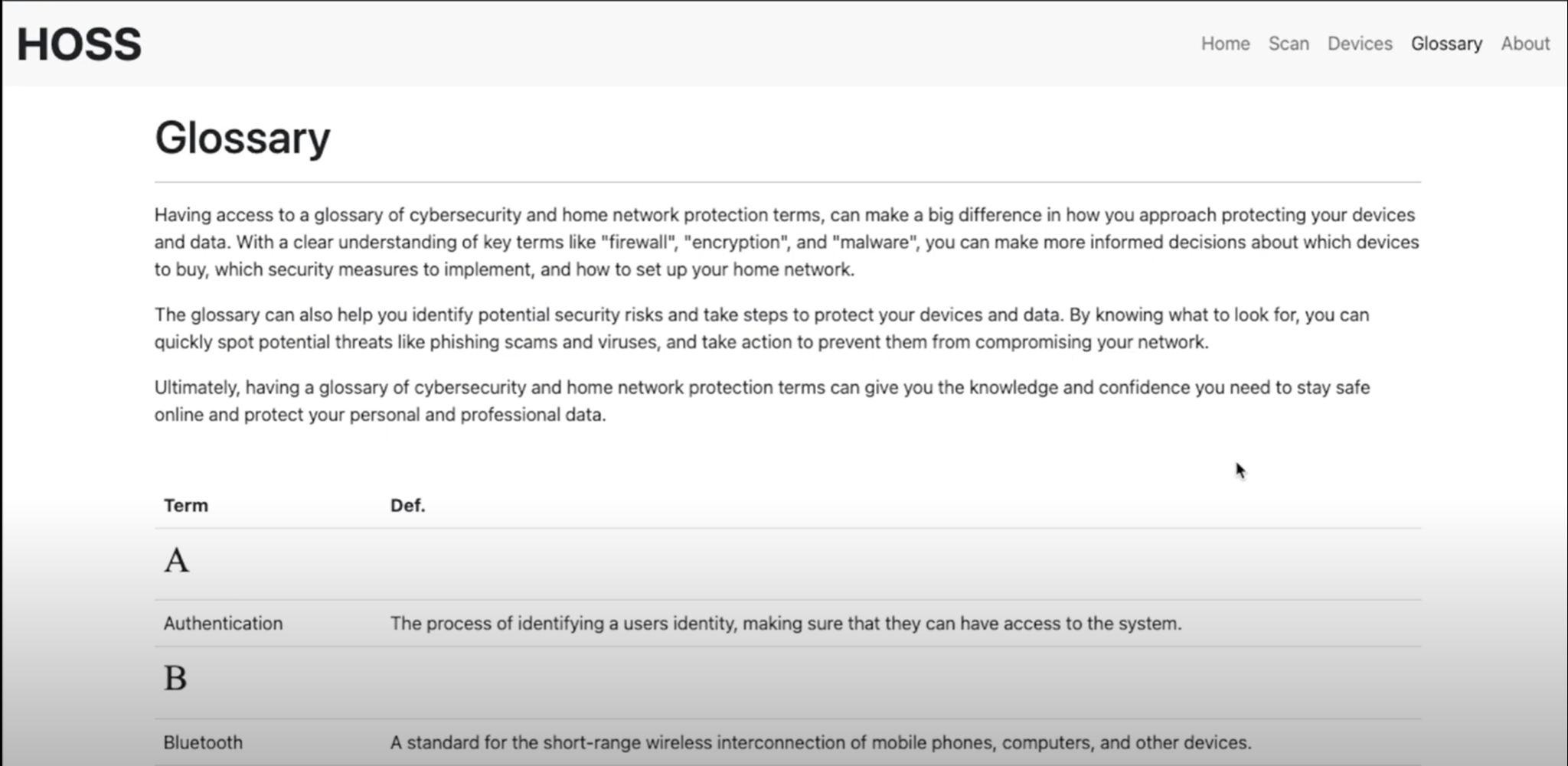
A User Interface was designed out of necessity as the point of contact between our User and HOSS. Below is a screenshot of the most important page, the Scan page. Here, the User shall click the “Scan My Network” button. After 10 to 15 seconds (depending on how many devices are in a User’s network), HOSS shall return a list of devices with their respective security scores and privacy scores. The User may also click on each returned device to view a dropdown of the device’s IP address, MAC address, Host name, and device type.

****

Another pertinent part of the User interface is the “Devices” page, which will show a list of all past devices scanned and detected in a User’s network. Any device detected in a scan shall be stored on the devices page.



The final pertinent page to discuss is the glossary. Since we cannot assume that all of our Users will be familiar with cybersecurity terminology, we have included a glossary page which goes over typical cybersecurity terms. Below is a partial snippet of our glossary.



**3.3.4. Scoring Module**

The Scoring Module incorporates the use of a CVE API as well as data about a device's security features from our MongoDB. A score for a brand is generated based on a weighted average of the severity of past vulnerabilities as well as if they sell your data. Then the score for a device is generated based on it's security features and that score is then curved based on the brands score. The same also applies to the privacy score which is calculated based on the data the device has from the user.

**4. Conclusions:**

**4.1. Results:**

Our team has largely succeeded in our goals for this project, but there are still some aspects which failed at being ideally implemented. As discussed earlier, we have APIs which automatically return a device’s model, MAC address, type, hostname, and prior CVE history. This information is seamlessly captured by our implemented framework and stored into our database for future use. However, our goal was to automate every feature that we used in scoring a device. Encryption method, authentication method, and default settings are pertinent attributes for our scoring module to consider, but our team was unable to implement neither an API, nor any other programmatic method for HOSS to automatically detect said features. Thus, our team must always manually research these features for each IoT device and input them into our database accordingly for future use. Since there are many IoT devices, this can be very cumbersome since it involves considering monitoring our MongoDB for devices which we do not have prior information on. However, we also understand that this is a project that is ambitious in scope due to the sheer amount of IoT devices it is expected to evaluate, and we will continuously research all new device’s features manually as they come via scans done by our Users.

This manual research is also the case for our privacy attributes. As discussed earlier, our privacy score is evaluated by if a brand sells the User’s data or not, and what kind of sensitive information a device contains (Social Security number, payment information, address, date of birth). After heavy research, we have not discovered any APIs or methods to automatically detect these attributes once we acquire a device’s brand and model, and thus have to resort to manually researching and inputting this information for each device. However, these aspects that we failed to automate still function as intended in our application. We simply hope to automate them in the future in order to minimize the constant maintenance and updating required by our server-side.

**4.2. Future:**

The strongest aspect of HOSS is that despite us failing to automate all the aforementioned tasks, we have built a very strong foundation for the future. The framework and code of HOSS is built in a boiler-plate fashion and is very welcoming and easy to read for the purpose of incorporating future updates. Thus, while we have a server-side team that will constantly update devices without any historical information in our database with all the aforementioned attributes required for scoring, we still have future plans for optimization and improvement. These plans but include, but are not limited to:

* Discovering an API which shall automate returning a device’s encryption method, default settings, and authentication method. This will solve the issue of our back-end constantly having to manually research and input these attributes for new IoT devices in our database which do not have this information previously.
* Discovering an API which shall automate returning a device’s information regarding if the brand sells the User’s data. Similarly, we’d like to discover an API which shall automate notating what type sensitive information a device stores about the User.
* Include another scoring criteria for security which involves detecting how much server traffic goes in and out of each User’s device. This was something we discussed in the planning stages, but were unable to implement due to the timeframe.
* Add an authentication method, providing our User the option to require a password for logging into their software.
* Provide users with recommendations on how to improve their security and privacy, along with specifying on the Scan page if a brand sells the User’s data or not, rather than simply factoring it into our privacy score. Currently, we factor this aspect into our privacy score, but do not specify to the User if the brand does or does not sell your data.

**5. References:**

NMAP API Website - <https://nmap.org/book/nse-api.html>

Fing API Documentation - <https://www.fing.com/images/uploads/general/Fing_Cloud_API_v1.0.pdf>

CVE API Website - <https://nvd.nist.gov/developers/vulnerabilities>

MongoDB Documentation - <https://www.mongodb.com/docs/>