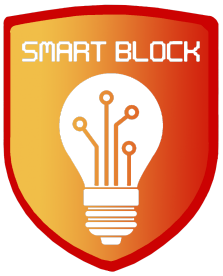


SmartBlock

Web Attack Tracking Software

Final Report

Team: sdmay21-17
Client: Cylosoft
Advisor: Doug Jacobson

Team Members:
Andrew Marek - Software Engineer / Administrator
Jamie Sampson - Security Engineer
Paul Degnan - Automation Engineer
Emily Young - Database Engineer
Megan Hill - Website Engineer

Team email: sdmay21-17@iastate.edu
Team website: https://sdmay21-17.sd.ece.iastate.edu/

Executive Summary

Background

Cylosoft is a custom website and IT consulting company located in Ames, Iowa. It has been found that many of their websites receive loads of malicious traffic. This traffic is recorded daily in the form of log files that are located in each website's respective directory. The bad traffic is usually detected by a human manually parsing the log files and looking at attributes, such as the location of the connecting IP address, and frequency of connections. Due to the arduous task of parsing the log files by hand, Cylosoft has requested a program to automate the process of blocking particular IPs that they deem malicious, a way to visualize the connections that are blocked, and a way to store the blocked IPs. SmartBlock is the solution that we created to address these requests.

Project Description

SmartBlock is a web-attack tracking application that aims to provide Cylosoft with a reliable and secure interface to gauge how and when their websites are being attacked. Notably, Cylosoft uses Microsoft Internet Information Services (IIS) to host their customer's websites. The project's goal is to secure Cylosoft's websites with appropriate Microsoft IIS settings to block potentially malicious connections and parse the IIS-generated logs to store and display information relevant to the administration of the websites.

Evolution

From the inception of SmartBlock, the overarching goal has always been the same: find a way to secure Cylosoft's websites in a programmatic fashion using technologies familiar to Cylosoft. Generally, our design shifted over the course of the year from writing a log parser to block the IPs ourselves to configuring Microsoft IIS to block them for us. Additionally, we decided to create a visual component in the form of a website for our client, such that the statistics regarding the web attacks are visualized and easy to see.

Our original intention of implementing our own program to block web-attacks shifted into configuring Microsoft IIS appropriately and recording the particularities of the log entries that IIS produces. From the beginning we created a web-attack blocking application that runs as a daemon on their servers, which would parse IIS log files as they come in - a daily process. The program would look for particular attributes, namely: IP addresses known to be malicious, addresses from uncommon locations, and addresses that connected too frequently to the websites under scrutiny.

We planned on writing a log parser in the form of a .NET console application that would look for IP addresses to block, using the aforementioned criteria. The program would then tell

Microsoft IIS to block the IP addresses, as well as marshal this information into objects to store in our database.

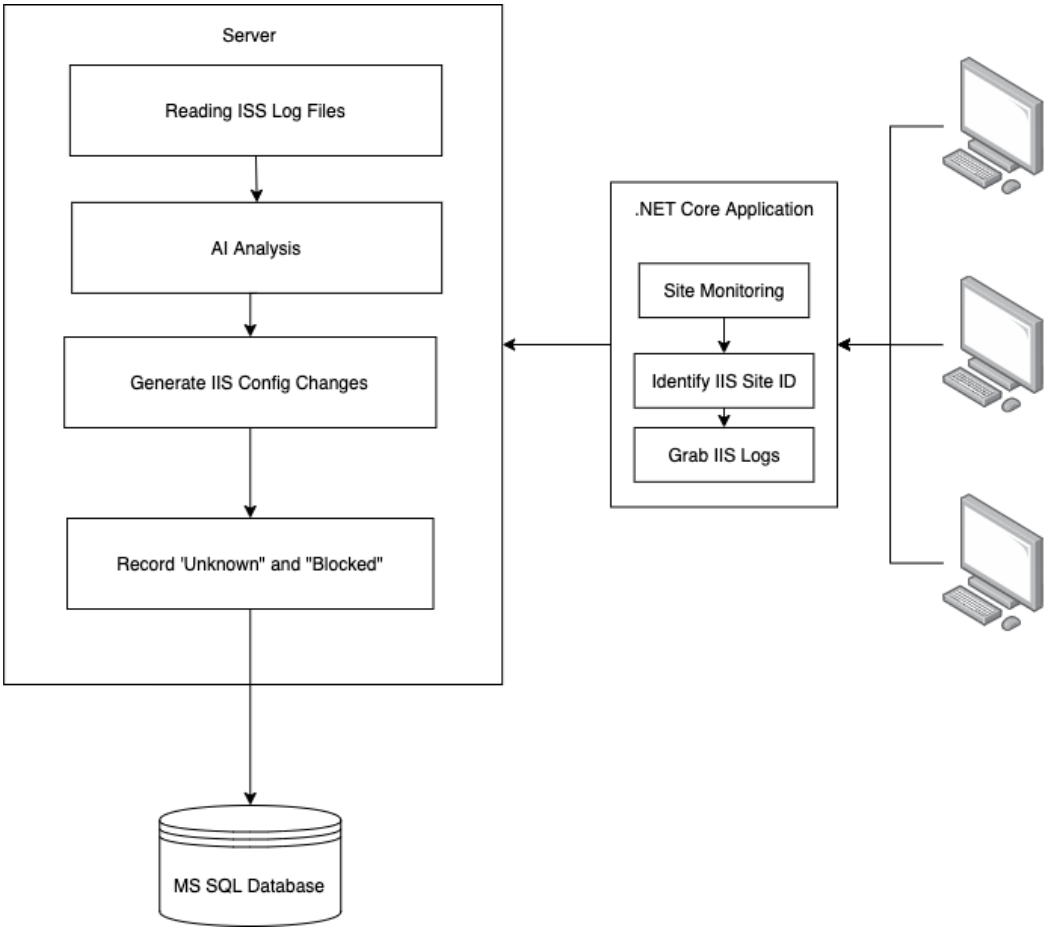


Figure 1 - Our initial module design of the SmartBlock system.

After exploring Microsoft IIS more extensively, we found that we could utilize Microsoft IIS's configuration settings to block connections, as we learned that IIS offered a configuration to perform similar functions natively. The advantage of utilizing IIS's native function has the potential to be far more reliable as Microsoft's products are battle-tested and there is no need to recreate the wheel. Thus, our goal shifted to automating the configuration of IIS and visualizing traffic information.

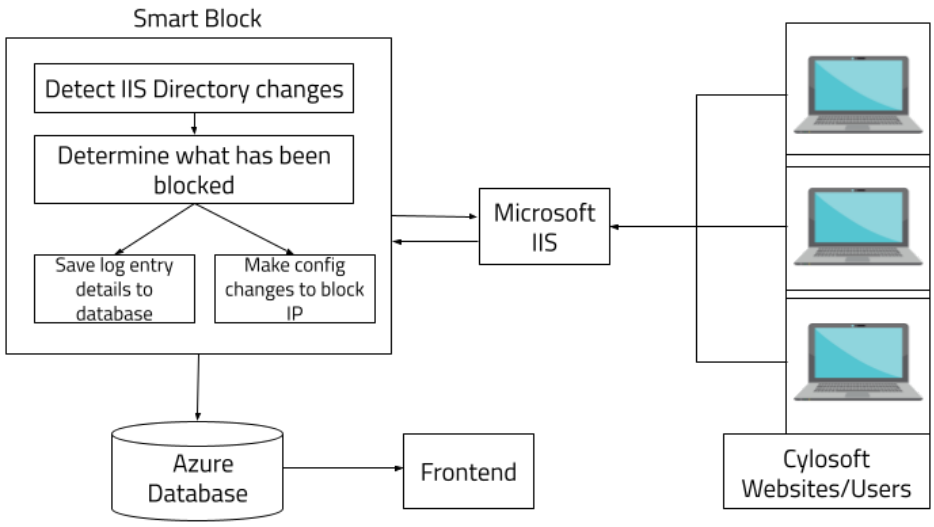


Figure 2 - Our final component diagram. Notably including a frontend that reads from our Azure database.

As one can see in Figure 2.02, we now introduce a front-end to the design and interface with Microsoft IIS more closely. These new design choices allow us to use Microsoft IIS to block the websites that are specified in the provided configuration files.

Functional Requirements

- Application should recognize bad web traffic based on invalid website URLs
- Application should identify bad web traffic exceeding a certain frequency
- User should be able to customize website URLs and frequency
- Application should block bad web traffic through IIS config changes
- Application should be able to read Microsoft IIS log files
- Application should allow configuration to IIS Site ID
- Application should keep a record of IPs that have been blocked
- Application should process multiple sites on a single server
- Application should protect the security of the user data collected
- Application database queries will have SQL injection protection

Non-Functional Requirements

Economic Requirements

- The design will take no longer than 500 person-hours
- Application development should take approximately 1000 person-hours
- Application database should not cost more than $100

Environmental Requirements

- Application should run on a Windows Machine
- Application should use an Azure Database
- Application should use .NET Core
- Application should use Github as its version control system

UI Requirements

- Application should produce verbose logging information
- Application should have detailed comments, readmes, and developer files
- Should provide a web user interface to view data about blocked logs

Standards

The bulk of our standards are best practices surrounding the operation of Microsoft IIS, writing standardized C# code that is compliant with Cylosoft's technologies, and coding practices. With that being said, we also wanted to follow Internet Standards. Below is a comprehensive list of guidelines we followed during the development of SmartBlock:

- Internet Standards - IETF
    - A set of standards that describe how the internet is put together that is accepted by the premier internet standards organization [1].
- Microsoft IIS best practices
    - Utilizing site dynamic security configurations
- Follow C# coding standards
- Develop using Cylosoft's technology stack

Engineering Constraints

- Application must run on Windows OS
- Application must use the client provided Azure database
- Application must use Microsoft IIS for blocking
- Application must be built using .NET Core framework to integrate with existing software
- Application code will be stored in a private Github repository

Security Concerns and Countermeasures

Cybersecurity

Our application focuses heavily on cybersecurity. We are focusing on two main cybersecurity threats. First, we are focusing on users that are hitting URLs that do not exist. Secondly, we are configuring IIS to handle any URLs that have already been deemed malicious. Cylosoft has provided a list of URLs that they frequently see in attacks. This list included details such as a CloudFlare connecting IP of “phpliteadmin.php” for websites that do not use PHP or “wp-admin” for websites that are not WordPress sites. Another sign of a malicious user accessing a website is, when a user accesses websites faster than humanly possible. This type of behavior implies that the users are likely bots or malicious users, as opposed to entities that should be accessing the website.

We allow users of our application to set how many URLs a user can be on at a single time and how many requests one can make in a given time period. This allows us to reduce the likelihood of false positives on websites that are heavy on redirects or could have a user clicking through a website very quickly.

Access to our database is controlled by IP. This means that in order to connect to the database, you need to both know all of the secrets necessary to connect to the database and be using an IP that has been given access to it. We also ensured that our codebase was set up in a private GitHub repo so that only necessary people would have access to our code.

Implementation Details

Internet Information Services (IIS)

Microsoft IIS is doing the majority of the work for our application in relation to blocking URLs. We allow users to contribute JSON config files for each site. Our application takes these configuration files and uses them to configure each site’s IIS settings. This makes SmartBlock highly customizable, so different sites can block based on different features. We also offer a default configuration that can be used across all websites that one is monitoring if a SmartBlock user does not wish to have website-specific customization.

Figure 3 - An example of IIS configuration settings in JSON format.

Microsoft IIS has various modules that allow one to keep their site secure and regulate IP traffic. One module in particular is the *request filtering module* which allows one to add rules or filters to a website's security settings such that it blocks specific patterns. One of our application's main focuses is to allow a user to add strings that should be denied, giving the user the option to query URLs. The request filtering module also includes a list of URLs that should be blocked if incoming traffic contains the provided URL string. This was our initial design before we discovered that Microsoft IIS already has this built in - it just needs to be configured.

Based on the configuration, IIS blocks IPs as necessary and generates a log file for each website. If a connecting IP is already blocked, IIS will generate a 500-level HTTP status code (the *sc-status* field) if the IP has already been blocked. This information gets fed into the file watcher component of our system.

File Watcher

SmartBlock's second largest component is the host of *FileWatcher* objects we deploy to detect new log entries. SmartBlock uses the C# *FileSystemWatcher* class in order to monitor directories and actively create alerts upon any changes. We use this component to tell SmartBlock that a new log file has been created, deleted, or edited. In our implementation, we create a FileWatcher object for each folder that instantiates a *LogParser* object. When the FileWatcher detects a change, update, or deletion in the FileWatcher's respective folder, the LogParser object parses through the log entries looking for any blocked IPs to place into objects and send to our database.

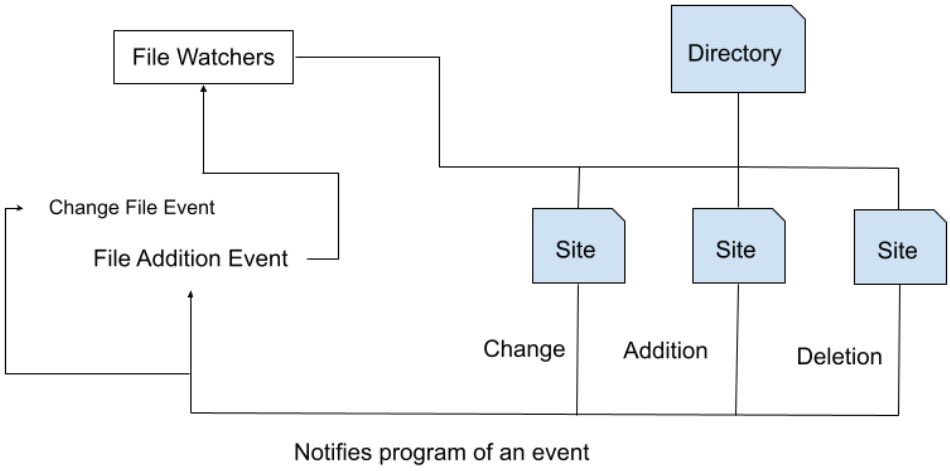


Figure 4 - A component diagram of the relationship between the file watchers and a nested directory hierarchy of websites.

Database

We are using an Azure database provided to us by Cylosoft. We have created a table in their database, *LogData*, which stores all of the information about which logs were blocked by IIS. SmartBlock is reading through IIS logs and checking which ones were blocked by our IIS settings. Those blocked logs are formatted and inserted into our database, where they can be preserved for posterity and accessed by our website.

Website

Along with an application to configure Microsoft IIS, we created a website to visualize data from the SmartBlock .NET application. Our application sends information about who is accessing Cylosoft's websites to the aforementioned Azure database, which is then displayed on our website.

Backend

Our website uses Hapi as the API framework. It uses an NPM package, *knex*, to connect to our database. The backend is set up to render the front end using the NextJS method *generateNextApp*, which allows our entire website to be built with one command since starting the backend was all that was necessary to render the frontend web pages. We provide two different backend endpoints. One of the endpoints provides data for the chart which includes website name, frequency, and date. The other endpoint provides more detailed information for

the table such as IP addresses csHost, csUriStem, and date/time. All API endpoints are prefixed with API, e.g., API/table-data and API/counts.

Frontend

Our website is built using the NextJS framework. For front-end UI components, we used Material UI to generate tables and any buttons that are used in the project. The table below is specifically a Material UI DataGrid, which gives us built-in functionality for filtering and sorting our data. We also used package recharts to generate the charts and the tooltip that shows the specific information for the graph. We use calls to the backend to populate the data for the chart and table.

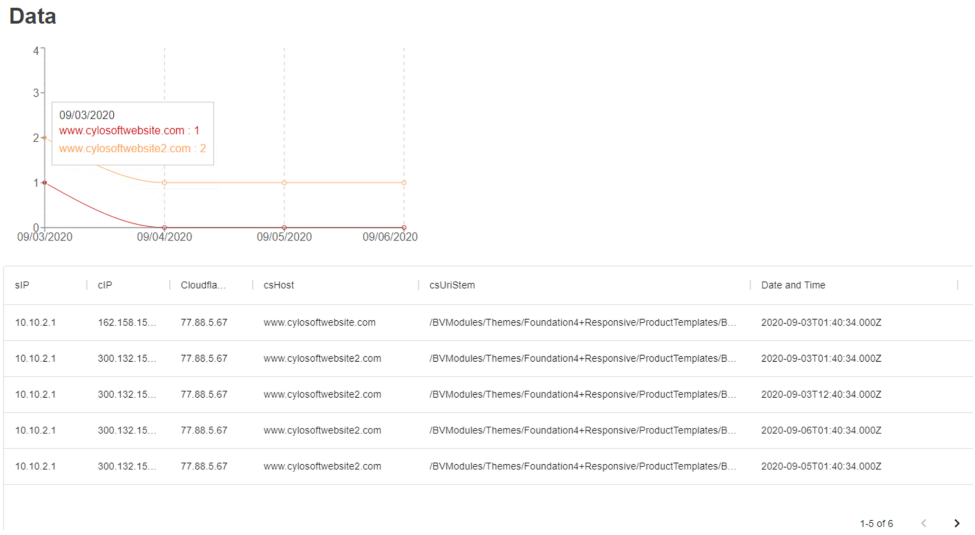


Figure 5 - The website component to SmartBlock visualizing the data received from the database.

Testing and results

Testing Application

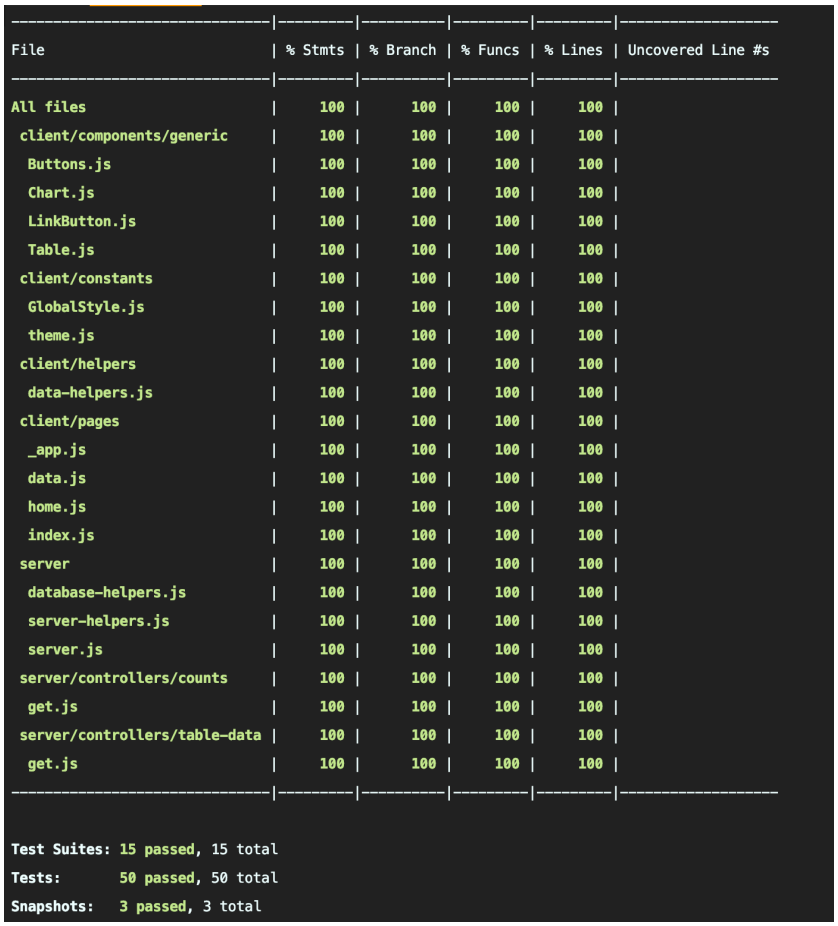
We took various steps in order to test that our application is working as intended. In order to get our application running and connected to the database and Microsoft IIS, we used Microsoft Remote Desktop. This allows us to set up a local website to test our IIS configurations. Additionally, we used the log-only mode in IIS to test our application in a production environment. Log-only mode allowed us to run our application without potentially blocking innocent users. It also checks that our application is behaving as expected, and is not blocking users unnecessarily or missing users that should have been blocked by the application.

Unit Testing Smart Block

In order to test our .NET Core application, we used MSTest which is a C# unit-testing framework written by Microsoft. Our tests focused on the LogParser's functionality. The tests ensured that as we updated our application, we did not accidentally break our existing functionality. We used example log files provided by our client for our unit tests so that our tests were representative of what would be seen in a live environment.

Unit Testing Website

In order to test our website we used Jest, a JavaScript testing framework that we chose due to its popularity and speed. Our unit tests consider both frontend and backend components of the website, testing features such as server and database connectivity, as well as the style of the frontend. Overall, the website tests provide 100% branch and statement coverage.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
client/components/generic	100	100	100	100	
Buttons.js	100	100	100	100	
Chart.js	100	100	100	100	
LinkButton.js	100	100	100	100	
Table.js	100	100	100	100	
client/constants	100	100	100	100	
GlobalStyle.js	100	100	100	100	
theme.js	100	100	100	100	
client/helpers	100	100	100	100	
data-helpers.js	100	100	100	100	
client/pages	100	100	100	100	
_app.js	100	100	100	100	
data.js	100	100	100	100	
home.js	100	100	100	100	
index.js	100	100	100	100	
server	100	100	100	100	
database-helpers.js	100	100	100	100	
server-helpers.js	100	100	100	100	
server.js	100	100	100	100	
server/controllers/counts	100	100	100	100	
get.js	100	100	100	100	
server/controllers/table-data	100	100	100	100	
get.js	100	100	100	100	

Test Suites: 15 passed, 15 total
Tests: 50 passed, 50 total
Snapshots: 3 passed, 3 total

Figure 6 - An example of our unit-tested web component.

Context

Related Works

Generally, there are many tools that help block cyber attacks and visualize the results that are available on the market. They each come with different and unique characteristics, with some even being open source. Some notable options include:

Snort - a very popular open-source intrusion prevention system. Snort uses a series of rules that define malicious network activity to detect packets that align with the rules [2]. Snort will then generate alerts for users. This is very similar to what we want to create, except we would want to make this specific to Cylosoft and also more "user friendly". A disadvantage of Snort is that Snort is mainly a command line utility, and we wanted our system to have a GUI component. It is also not very "user-friendly" in the sense that one needs to make rules in an oddly-formatted configuration file and also have experience using command line tools. However, advantageously it is very lightweight, robust, and fast - which makes it extremely appealing.

Splunk - Splunk is a software for searching, monitoring, and analyzing machine-generated data [3]. This concept could very well expand to our use case, for instance, analyzing Microsoft IIS logs. One could use it to create dashboards that aid in monitoring events in network activity. One huge advantage of Splunk is that it is community driven, where they have a host of apps and add-ons for Splunk which can increase its usefulness. A downside to Splunk is that it may have a steep learning curve and supposedly some users report that building queries can be cumbersome.

Datadog - Datadog is a monitoring service for cloud applications and servers. Datadog has a dashboard, alerting, and visual metrics to describe what is happening in the parsed log files [4]. Some advantages of datadog is that you can describe the format of the log that you want to parse - making it versatile. A disadvantage of datadog is that it's not open source, which isn't too impactful, but open-source projects are generally more modifiable and transparent.

Overall, there are probably countless technologies that do something similar to what we have created, however what may differentiate our product from others is that we want to simplify the process and setup of IIS and visualize it with modern technologies. Another important point is that our product is specific to Cylosoft's setup. The task at hand was to integrate our program with Microsoft IIS as well as possible, which naturally called for a custom .NET application as opposed to the aforementioned products.

References

[1] “Internet standards,” *IETF*. [Online]. Available: https://www.ietf.org/standards/. [Accessed: 20-Apr-2021].

[2] “New to Snort?,” *Snort*. [Online]. Available: https://www.snort.org/. [Accessed: 25-Oct-2020].

[3] “The Data-to-Everything Platform Built for the Cloud,” *Splunk*. [Online]. Available: https://www.splunk.com/. [Accessed: 25-Oct-2020].

[4] Datadog, “Datadog,” *Cloud Monitoring as a Service*, 14-Jul-2016. [Online]. Available: https://www.datadoghq.com/. [Accessed: 25-Oct-2020].

Appendices

Appendix I

Dependencies

- Website component
    - NodeJS
    - NPM packages
- .NET core
- NLog

Operation Manual

Application:

The source code for SmartBlock is available at its official GitHub repository: https://github.com/JamieSampson08/sdmay21-17. One should note that this repository is private, and this application is a product of Cylosoft’s. Thus, one must obtain permission before accessing this repository. However, assuming one has access to the program, the following steps are used to configure and run the software:

- Install the required dependencies listed above using your preferred method.
- Download the code by cloning the repository or downloading it manually from GitHub.

- To configure IIS settings specific to your website, navigate to the *SiteSettings* folder and create a new .json file which will specify the types of filterings and dynamic security settings you'd like to use. Multiple templates have been provided in the same folder.
- Edit the Sites.json file, which contains a list of sites that will be taking on specific configurations. Enter the names of the JSON settings files that you previously created.
- Lastly, if you have a folder path different from the default setup that we have created, i.e., not in the program folder, one can insert the relative path of the directory containing the sites log files in our Constants.cs file by changing the *BaseLogFilePath*.
- After one has followed the previous steps, one can build the program and our application will begin monitoring the specified folder(s) for any changes.
- Once a change is detected, SmartBlock will alert the user via a logger, and also a visualization on our website component if launched as well (see below for a setup guide).

Website

Follow these steps to run the website component of SmartBlock:

- Clone or fork the repository from GitHub after obtaining access.
- In your terminal, use the *cd* command to navigate into the sdmay17-21-website folder
- In your terminal enter:
    - *npm i*
    - *npm run dev*.
        - This will install the necessary npm packages and launch the website locally.

The website will run at 0.0.0.0:3000. You can go to 0.0.0.0:3000/data to see the page that we have shown screenshots of in other parts of our report.

Appendix II

Alternative / Initial Design:

Our initial design ended up being reasonably close to what we ended up creating. The only major change from our design at the end of last semester is that instead of sending the IPs that are going to be blocked to IIS, we are letting IIS enforce the blocking rules. We are then reading from the logs which IPs that IIS blocked and creating entries in our database, eventually displaying them on our website. If one wanted to perhaps deviate from the capabilities of Microsoft IIS, or use another website hosting system, one could add on to this program and develop new criteria for blocking IPs. Once the new criteria is developed, one could utilize the API Microsoft provides to interface with IIS.

Appendix III

Code

Since our code is owned by Cylosoft, we are not going to share it in this document. If you believe you need to see the code reach out to Andrew Dakin at Cylosoft to obtain his permission. It is stored in a GitHub repository at this URL: https://github.com/JamieSampson08/sdmay21-17, but the repository is private, so permission must be obtained before seeing the code