

Team: sdmay21-17

Andrew Dakin (Client)

Douglas W. Jacobson (Advisor)

Andrew Marek (Lead Engineer)

Paul Degnan (Administration)

Jamie Sampson (Test Engineer)

Megan Hill (Task Manager)

Emily Young (Product Owner)

sdmay21-17@iastate.edu

https://sdmay21-17.sd.ece.iastate.edu

Revised: Nov. 15th (V3)

Smart Block

Webserver Attack Blocking AI

Design Document

# Executive Summary

Development Standards & Practices Used

Our project is strictly software, so no hardware will be used. We will be following a loose version of agile development and object-oriented programming. Part of our testing process will use end-user acceptance testing. Architecturally, our code will be using REST APIs and implementing good practices around database security.

Summary of Requirements

* Identify malicious IP addresses by analyzing IIS log files
* Use AI to block malicious IP addresses and store that data
* Create a .NET Core Console application that runs in Windows

Applicable Courses from Iowa State University Curriculum

* SE 329 - Project management
* COM S 363 - SQL database knowledge
* COM S 309 - Software Development Practices
* S E 339 - Software Architecture and Design

New Skills/Knowledge acquired that was not taught in courses

* .NET Core
* IP address blocking
* IIS logs
* Identifying malicious attacks (Cyber Security)

Table of Contents

[1.4](#_heading=h.1t3h5sf) Requirements 4

[1.5](#_heading=h.4d34og8) Intended Users and Uses 4

[1.6](#_heading=h.2s8eyo1) Assumptions and Limitations 5

[1.7](#_heading=h.17dp8vu) Expected End Product and Deliverables 5

[2](#_heading=h.3rdcrjn) Project Plan 5

[2.1 Task Decomposition 5](#_heading=h.26in1rg)

[2.2 Risks And Risk Management/Mitigation 6](#_heading=h.lnxbz9)

[2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria 6](#_heading=h.35nkun2)

[2.4 Project Timeline/Schedule 6](#_heading=h.1ksv4uv)

[2.5 Project Tracking Procedures 6](#_heading=h.44sinio)

[2.6 Personnel Effort Requirements 7](#_heading=h.2jxsxqh)

[2.7 Other Resource Requirements 7](#_heading=h.z337ya)

[2.8 Financial Requirements 7](#_heading=h.3j2qqm3)

[3 Design 7](#_heading=h.1y810tw)

[3.1 Previous Work And Literature 7](#_heading=h.4i7ojhp)

[3.2](#_heading=h.2xcytpi) Design Thinking 7

[3.3](#_heading=h.1ci93xb) Proposed Design 7

[3.4 Technology Considerations 8](#_heading=h.3whwml4)

[3.5 Design Analysis 8](#_heading=h.2bn6wsx)

[3.6](#_heading=h.qsh70q) Development Process 8

[3.7](#_heading=h.3as4poj) Design Plan 8

[4 Testing 9](#_heading=h.147n2zr)

[4.1](#_heading=h.3o7alnk) Unit Testing 9

[4.2](#_heading=h.23ckvvd) Interface Testing 9

[4.3](#_heading=h.ihv636) Acceptance Testing 9

[4.4](#_heading=h.32hioqz) Results 9

[5 Implementation 10](#_heading=h.1hmsyys)

[6 Closing Material 10](#_heading=h.41mghml)

[6.1 Conclusion 10](#_heading=h.2grqrue)

[6.2 References 10](#_heading=h.vx1227)

[6.3 Appendices 10](#_heading=h.3fwokq0)

List of figures/tables/symbols/definitions

Figure 2.4.1 Semester 1 Gantt Chart

Figure 2.4.2 Semester 2 Gantt Chart

Figure 3.7.1 Module Diagram

# 1 Introduction

## Acknowledgement

Cylosoft, and particularly Andrew Dakin, has provided us with significant resources necessary to complete this project including, sample logs, and technical experience. We have also received help from our ISU advisor, Doug Jacobson.

## Problem and Project Statement

Cylosoft has a CMS Platform used to host many customer websites. On a regular basis, the websites are probed and tested for issues by bots and hackers. SmartBlock is written to allow the website to actively detect and stop these attacks. This involves writing an application to read web server traffic logs in real-time. The logs are scanned by artificial intelligence and used to block bad traffic. The AI can peer across all the CMS sites and then write IP blocking rules to eliminate the attackers attempts.

## Operational Environment

Our product is expected to be able to handle multiple sites on a single server functioning on a Windows Machine. The data collected from our application will be held in an accessible database for Cylosofot to monitor. There are no physical components to our project, everything is digital.

## Requirements

Functional Requirements

* Application should recognize bad web traffic based on definitions provided by Cylosoft
* Application should block bad web traffic through IIS config changes
* Application should be able to read Microsoft IIS log files
* Application should keep a record of IPs that were blocked
* Application should process multiple sites on a single server
* Application should have a perceptive AI
* Application should protect the security of the user data collected
* Application database queries will have SQL injection protection

Economics 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 that would enable the application to become open source

## Intended Users and Uses

Our application’s main end user is to be our client, Cylosoft. Cylosoft intends to use our application to help them cut down on resources that are being held up by manually identifying and blocking bad web traffic that accesses their product. In a discussion with our client, it was decided that, based on the final deliverable, we may make our code open-source and available to anyone.

## Assumptions and Limitations

Assumptions

* Server provided to run the application on.
* Application will run in the background.
* AI will be able to be used by anyone, not just the company.
* Will be provided with access to real-time IIS logs.
* If used by other companies, their files are formatted the same as the company we are developing the application for.

Limitations

* Not checking for a spoofed location.
* Not checking for changed IP addresses.
* Will only block IP addresses based on suspicious activity shown in IIS logs, and cannot guarantee to prevent all attacks.
* Not testing integrity of data.

## Expected End Product and Deliverables

The primary goal of our project is to aid Cylosoft in preventing cyberattacks. This will manifest itself in a piece of software that will look through Cylosoft’s server logs, detect suspicious activity, and take appropriate action to prevent the source of the suspicious activity from accessing Cylosoft’s website. Thus, the main product that we will be producing for Cylosoft is a web-attack blocking AI software.

# Project Plan

## 2.1 Task Decomposition

Our project can be split up into two stages where each stage has their own tasks and subtasks.

Stage One: Create a .Net Console application that reads IIS log files

1. Create a MSSQL database to host the storage of log data
   1. Gather sample log files to create fields for the necessary pieces of a log file.
   2. Implement restricted user permissions to access database data.
   3. Would run on Azure SQL
2. Create a private GitHub repo for future open-source compatibility.
   1. Add any sensitive material and production code as we go here.
3. Create a .Net Core application
4. Connect the Azure database with the .Net Core application
   1. Enable the application to store new entries into the database.
5. Connect the .Net Core application with IIS logging to read in data
   1. Create a scalable algorithm to handle various fluxes of logs being received.
6. Determine what qualifies as a bad or good log file and the related signatures of each.

Stage Two: Design an AI that will block bad incoming traffic

1. Implement an AI that dynamically analyzes the validity of incoming traffic.
2. Block a specific IP address and record any identifying signatures to feed back into the AI.

## 2.2 Risks And Risk Management/Mitigation

1. Create an Azure database to host the storage of log data: 0.1.
   1. The risk for this step is 0.1 because our team is familiar with MSSQL database and it is a good reliable tool that we can easily use.
2. GitHub: 0.1
   1. The risk for this step is 0.1 because we have all used git before, and it is the version control system our team is the most familiar with.
3. .NET Core Application: 0.3
   1. The risk for this step is 0.3 as not everyone has experience with .NET console applications and there is a chance it doesn’t work.
4. Connect with IIS logs: 0.3
   1. The risk for this step is 0.3 because while we have not worked with IIS logs before, they should be fairly easy to connect to the .NET core application.
5. Implement AI: 0.4
   1. The risk for this step is 0.4 as the AI should be difficult to make and implement, but it is a necessary part of our project.
6. Block IP Addresses: 0.3
   1. The risk for this step is 0.3 because blocking an IP address should be fairly easy but there is a risk whoever was blocked just changes their IP address.

## 2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Criteria:

The two main criteria for our web attack blocking software will be accuracy and efficiency.

In terms of accuracy, it is important that our product is not missing any potential malicious actors that are attempting to access Cylosoft’s website. Therefore, one of the main metrics we will measure is attack classification correctness. This can be defined as the percentage of attacks we are classifying as malicious activity compared to the actual percentage of attacks that are malicious. This criterion is the main focus for the project.

The second criterion is how fast and efficient our software is detecting these attacks. Speed and efficiency are key pieces because if a malicious actor is attempting to access a site, one would not want the actor performing potentially dangerous activity while waiting to be blocked. Thus, we will measure how long it takes for our AI to detect and block the actor in some standard time measurement such as milliseconds.

Metrics:

1. Attack classification correctness (% of correct classifications)
2. Speed / detection time (m/s)

Milestones:

1. Setup fundamental resources.
2. Ability to parse IIS logs and extract relevant information.
3. Detect an attack from IIS log successfully.
4. Implement AI using the algorithm to detect attack.
   1. Detect attack with >80% accuracy.
5. Block an IP address automatically after access is categorized based on extracted logs.
   1. Classification with >95% accuracy.
   2. Nearly instantaneous detection time.

## 2.4 Project Timeline/Schedule

For our scheduling plans, please refer to the documentation that is linked below. This link will take you to our [Gantt chart](https://docs.google.com/spreadsheets/d/1XKg0tS8S4s9vkntKFb-iq3ANwFSfIdAKgnCr7fy26fw/edit?usp=sharing) that covers both the fall 2020 and spring 2021 semesters.

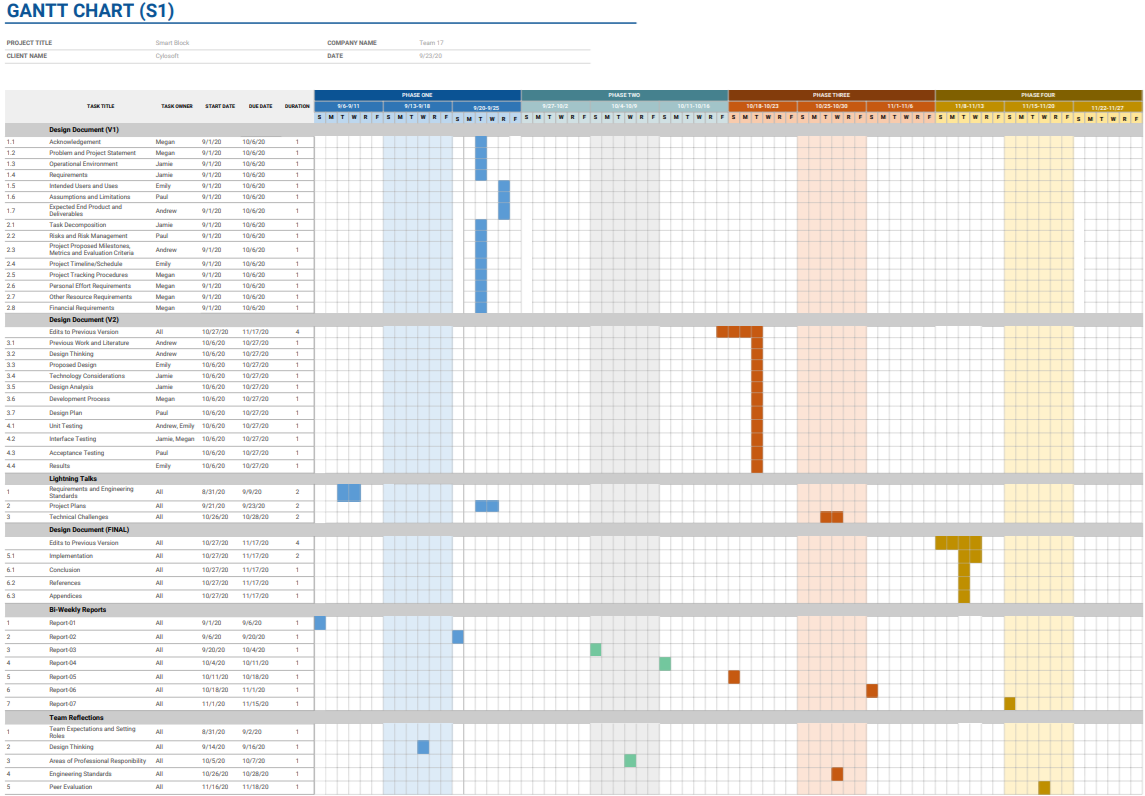


Figure 2.4.1 Semester 1 Gantt Chart

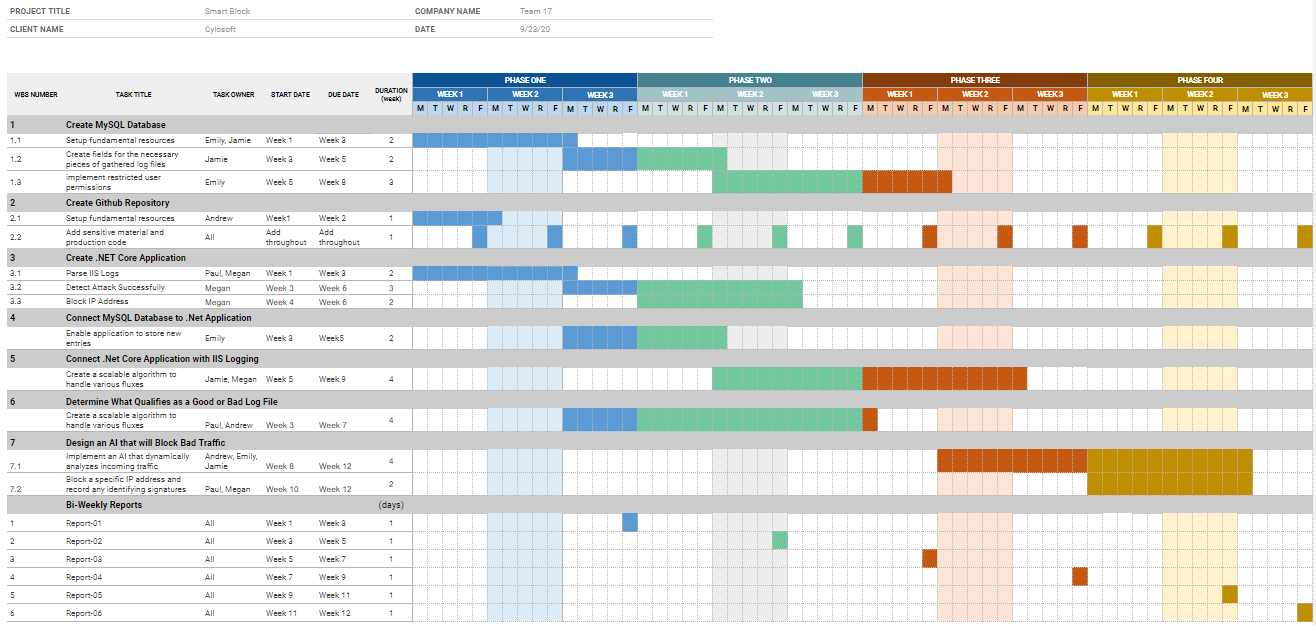


Figure 2.4.2 Semester 2 Gantt Chart

## 2.5 Project Tracking Procedures

Our team will be using GitHub to store all of our code for this project. We will use GitHub issues with labels for code tracking. This will allow us to keep track of everything that we need for the project in one place, while making it easier to link to existing code for clarifications. It will also allow us to keep track of why we are doing certain things (e.g., bugs, feature work, specific task). We will also be using Slack and WebEx to communicate with each other to define all of the necessary aspects of the project. This could include determining necessary features from the clients or breaking up work between the team members.

## 2.6 Personnel Effort Requirements

|  |  |
| --- | --- |
| Task | Hours |
| Create a Azure database to host the storage of log data | 15 |
| Gather sample log files to create fields for the necessary pieces of a log file. | 10 |
| Implement restricted user permissions to access database data. | 10 |
| Create a private GitHub repo for future open-source compatibility. | 5 |
| Create a .Net Core application using .NET 4.8 or 5.0 | 30 |
| Connect the Azure database with the .Net Core application | 35 |
| Enable the application to store new entries into the database | 30 |
| Connect the .Net Core application with IIS logging to read in data | 40 |
| Create a scalable algorithm to handle various fluxes of logs being received. | 75 |
| Determine what qualifies as a bad or good log file and the related signatures of each. | 25 |
| Implement an AI that dynamically analyzes the validity of incoming traffic. | 100 |
| Block a specific IP address and record any identifying signatures to feed back into the AI. | 25 |
| Total | 400 |

## 2.7 Other Resource Requirements

In order to complete the project, our team will need a variety of sample logs to test our application. These logs will need to have a combination of normal traffic and malicious traffic, with the malicious traffic marked (including why they are believed to be malicious) so that we know which scenarios to block. We will also need some database space so that we can keep a record of all logs that were blocked. Once the project is ready to be tested, we will need access to a dev environment so that we can test blocking traffic, without potentially negatively impacting the prod environment.

## 2.8 Financial Requirements

This project does not require any hardware, and we will be using all on prem databases, so there will not be any financial obligations for this project.

# 3 Design

## 3.1 Previous Work And Literature

A large part of our work for our web attack blocking AI will be reviewing best practices for log analysis of network logs. Our primary goal is to create a system that monitors network logs in real time to alert Cylosoft of an attack and potentially perform an action to stop an attack. Thus, taking advantage of literature regarding making such a product and looking at example code of something similar will be crucial. Some other literature we would want to review is best practices regarding development in C# - our main tool within our tech stack. One particular resource that would be of use is the documentation for C# on Microsoft’s website [1].

Generally, there are many “network” scanning tools available on the market. They each come with different and unique capabilities, 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 would want our system to eventually have a GUI. It is also not very “user-friendly” in the sense that one needs to make rules in a 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 want to accomplish with our web attack blocking AI. However, what may differentiate our product from others is that we want to simplify the process and setup. We also want to make our product unique for Cylosoft, however still make the code modular enough such that we are effectively “future-proofing” it.

## Design Thinking

The main objective of our design is to enable detection of malicious accesses to Cylosoft’s website. Some secondary objectives and non-functional requirements are ease of use and reusability. One could make an application that monitors logs for attacks and performs an appropriate respective action, but the interface for the application is important. This is something that will differentiate our AI from others; as noted above many of the applications created are somewhat difficult for new users to grow accustomed to. Hopefully with a more specific use-case in mind, being Cylosoft’s logs, it will be more trivial to implement. In terms of reusability as a goal, we acknowledge that technology is ever-changing, so in our *define* aspects, we want to make it easier to monitor different types of logs for the future.

Other design choices and considerations:

1. Reusability: writing very modular code and following best practices.
2. User-friendly: creating an interface so that users don’t have to learn a new skill to use the software.
3. Robust documentation: allows for easier debugging and facilitates user-friendliness.
4. Our choice of tech-stack: .NET - compliant with Cylosoft’s requirements.

## Proposed Design

The majority of the work that we have done so far has been on very small data sets, such as individual logs or only one or two logs at a time. We have been able to implement parsing individual logs, figured out how to send data to a database, and implement a basic AI that recognizes some of the most common attack vectors.

Our final project will be a .NET console application. This application will read IIS logs that come from a variety of Cylosoft sites. It will then use information such as IP, uri, query, method and status to determine which of the logs are coming from requests made by bots or other possibly malicious activity using artificial intelligence and the criteria provided, such as common attempting to hit WordPress URLs on non-WordPress sites, receiving more requests than a person could generate, etc. Once the logs that are coming from bots are identified, we will block the IP addresses associated with those logs by making IIS configuration changes. It will also log any blocked IP addresses to a database, using Azure SQL, so that there is a record of all blocked IP addresses.

This design satisfies some specific constraints that we were given by our client, including that the application is a .NET console application, and that it uses Azure SQL.

We will also be following all the necessary security standards to ensure that any data that we gather from the IP logs is kept secure. This will include following rigorous security precautions. One standard that we will attempt to follow is PCI DDS. While PCI is intended to keep credit card information data secure, those same principles can be used on this data as well, even though it does not contain credit card information. This also means that we are keeping our code in a private repository to ensure that no one has access to the code and could use it to reverse engineer an attack that would not be blocked.

This proposed design satisfies the functional requirements described in section 1.4

## 3.4 Technology Considerations

Many of the technologies we will use were given to us as design constraints, so changes are not allowed meaning design alternatives cannot be considered.

.NET Core

* Strengths: Matches rest of Cylosoft infrastructure, fits well into Microsoft environment, efficient performance, cross platform compatibility
* Weaknesses: Our team is less familiar with it then some other frameworks, some development tools for it are not cross-platform compatible

MSSQL

* Strengths: Easy integration into Microsoft Environment, matches existing infrastructure, similar to other SQL type languages, feature rich
* Weaknesses: most team members are more familiar with other SQL variations, more expensive than some other databases

Microsoft IIS:

* Strengths: Easy to integrate into Microsoft Environment, matches existing infrastructure, allows programmatic configuration changes, easy to control
* Weaknesses: Windows specific (Mac users cannot run)

Possible Solutions:

Our two main problems are:

* Lack of Experience: The main solutions are to use the technologies more so that we become more familiar with them, this is something that will happen over time. Another solution is to do code reviews, so that people with more experience can comment on how to address the best practices and make sure that the correct procedures for everything are still being followed
* Incompatible Technology: Many members of our team use Macs and not all Microsoft technology is compatible. In recent years Microsoft has been coming out with more and more technology for Macs making this feasible, but some testing will still need to be done on virtual machines (VirtualBox, Parallels, etc.).

## 3.5 Design Analysis

Our team is still in the building process that is implementing the design described in section 3.3. That being said, we do not have an accurate response as to whether or not our proposed design is sufficient. However, in a lot of our test .Net applications, we have successfully been parsing our IIS Logs that our client has provided us. So, the first major portion of our proposition, has a very high likelihood of success. Since most of our team is not proficient with some of the technology, I am positive that there will be some slight changes to come to our design. This would potentially be due to our team assuming that a function within the build process would work when, in reality, these features are not compatible with this technology. Of course, this will only be proven throughout our continued work with Cylosoft.

## Development Process

Our team has chosen to use the Agile approach to develop our product. Since most of our team is strictly remote, we have agreed that this would help improve our team’s communication and productivity. The main reason that Agile works for our team is that many parts of our development tasks rely heavily on specific portions of the project to be most or fully functioning to continue making progress towards our completion of the project. We believe that completing tasks in sprints would be the smoothest option to keep all members of the team busy while also keeping the members from feeling overwhelmed. We also wanted the chance to have multiple people working on the same tasks with each other to help alleviate the stress from the areas where there is a lack of experience. Agile gives us the opportunity to offset certain tasks so that there can be more team collaboration.

## Design Plan

As part of our design, one of our requirements is to be able to monitor multiple sites at the same time on the same server, as depicted in figure 3.7.1. We are also addressing the use-case of needing the application to store data on the sites that were blocked through using a MS SQL database. Another part of our design is to have the .NET Core application handle the work of getting the IIS log files and managing what sites are being monitored. Our server will host APIs that analyze the logs and take the necessary action depending on the results. Our server will also support the AI and make requests to the MS SQL database.

## 

Figure 3.7.1 Module Diagram

# 4 Testing

## Unit Testing

Unit tests will be performed on the following aspects of our software application.

* ability to read in all necessary data fields from log files
  + correct data extrapolated
  + incorrect expected data throws an error
  + no data for a field is handled
* ability to store data to database
  + successfully creates an entry to the database
  + handles attempts to store invalid data
  + handles attempt to store no data
* ability to block bad IPs and not block good IPs
  + bad IP is blocked
  + good IP is not blocked
  + unknown IPs are recorded
* ability for all flags on bad IPs to be caught by AI
  + each flag tested individually results in an analysis to be blocked
  + IP that doesn’t fit any of the flags is accepted
* ability to have different types of access to database functions
  + x username and password give master access to database
  + x username and password give read access to database
  + x username and password not given access to database
* ability to restrict access to raw application code upon exporting application
  + admins have full visibility
  + non-admins have no visibility

## Interface Testing

* Test the communication between the IIS logs and the console application
  + Test that the application can connect to the IIS log.
  + To make sure the application is connected to the IIS logs, take the first word in it as an input and make sure it is returned.
* Test the ability for console application to be run
  + When the web application is opened, test the connection to the console application before doing anything else.
  + If it is not connected to the console application, show an error message.
* Ability for the console application to connect to the database
  + Add items to the database: Create a function that creates a test object in the database to verify that the database and console application are connected.
  + Get items from the database: Make sure we receive information before trying to add or do anything with that information.

## Acceptance Testing

* Create a test that blocks an IP address to show that the console application will fulfill its functionality in blocking an IP address.
  + This is the main functionality so creating a test that we can run at any time to show the blocking of an IP address is important.
  + Use a fake IP address/machine to show that it can no longer access the website after being blocked
* Create a test that either shows an error message or shows the console application in a web application.
  + Show an error message if the console application errors out.
  + Show the console application if no errors.
* Create a test that shows the application is reading and determining whether the traffic is malicious or not for all traffic.
  + Go through each piece of traffic being looked at by the console application and show whether or not the application determines it as malicious to show that no traffic is being missed or erroring out.
* Create tests for all known edge cases to make sure the console application will not error out.
  + When edge cases are found, create tests for them to show that there will not be an error while running the console application for a long period of time.

## Results

* Acquired Azure database information from Cylosoft and connected to it from our home computers through DataGrip.
* Wrote an SQL script to create tables and add dummy values
* Parsed log data into SQL value format and confirmed that the formatted output can be directly applied into an insert SQL command.
* Created a FileWatcher that monitors changes made to a specific file. Ensured that it picks up new data and parses it.
* Successfully created a class to parse a log entry and split into its corresponding entry values.
* Created the ability for a file to be read from the console application when the FileWatcher catches a change to the IIS Log.

# 5 Implementation

At the end of this semester, we will have the following things:

* IIS log file parsing code to extract relevant data
* Setup for an Azure database
* A .NET Core application
* Connection within application to Azure database
* Checks for common malicious signs

These things set up the foundation for modulating our project as seen in our proposed design. Our plan for next semester are the following:

* Connect our application with our client’s specific database
* Acquire more log files to identify malicious behaviors
* Implement and organize any new malicious signs to our AI
* Store relevant information of blocked IPs into the database
* Block identified malicious IPs
* Document for open source
* Connect IIS log parsing to actual IIS log data stream

Our stretch goal, not required features, is to create a UI for the information we gather and to create a more complex system of blocking malicious IPs.

# 6 Closing Material

## 6.1 Conclusion

So far in our project we have laid out our project plan, designed the application, prepared our test cases, come up with a plan for implementation, and started creating the application. We have figured out how to tell when a log has been updated, open and read the file from the application, created some basic AI, set up the database infrastructure, connected to the database, and set up a .NET console application’s core infrastructure. Our goal is to have a working .NET console application that can use AI to block IP addresses that create suspicious activity. This suspicious activity can be seen on the IIS log files that our application will parse through, flag, and block the necessary IP address. To achieve our goals, we will be following our project plan including the designs we came up with, testing we have thought out, and implementation guidelines. This is the best plan of action as we have spent a lot of time making sure the creation of our project will be done to the best of our abilities. By following the design guidelines everyone on our team will be on the same page and know what they need to accomplish. By following the testing guidelines any problems in our application can be caught and fixed. By following the implementation guidelines, we will implement our project with the highest quality and efficiency we can. We have thought out many solutions, but this project plan lays out the best possible course of action we can take.

## 6.2 References

[1] B. Wagner, “C# docs - get started, tutorials, reference.,” *C# docs - get started, tutorials, reference. | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. [Accessed: 25-Oct-2020].

[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].

## 6.3 Appendices