Cloudflare WAF Scanner

Final Report

SDMAY21-16
Client: Cylosoft
Advisors: Dr. Yong Guan, Dr. Berk Gulmezoglu

Ben Feddersen: Stakeholder Correspondent
Eric Reuss: Documentation Manager
Giovanni Mejia: Report Manager
Jordan Heim: Documentation Manager
Presiian Iskrenov: Meeting Scribe
Ryan Burgett: Meeting Leader

sdmay21-16@iastate.edu
http://sdmay21-16.sd.ece.iastate.edu

Revised: April 25, 2021

Executive Summary

Development Standards & Practices Used

- Agile
- REST
- TCP/IP
- HTTP

Engineering Standards Used

- IEEE 1028-1997 Standard for Software Unit Testing
- IEEE 12207-2017 Software Life Cycle Process
- IEEE 16326-2009 Project Management

Summary of Requirements

Microsoft ISS will generate text-based log files of each web request. A console-based application will be created to find the current log file for the site that is being monitored. The application will then send the log file to the web service to be stored and processed. A database scanning application will be created to monitor the log files and identify suspicious web activity using the Cloudflare API. When suspicious activity is detected on the server, the potentially dangerous IP will be blocked or challenged for twenty-four hours. A web application will also be created to view the contents of the database

Applicable Courses from Iowa State University Curriculum

- COM S 227
- COM S 228
- COM S 252
- COM S 309
- COM S 311
- COM S 321
- COM S 329
- COM S 339
- COM S 352
- COM S 363
- COM S 474
- CPR E 430
- CPR E 431

New Skills/Knowledge acquired that was not taught in courses

- .NET framework
- Telerik and Kendo UIs
- IIS log files and general web traffic moderation
- C#

Table of Contents

List of figures/tables/symbols/definitions

- Definitions:
  - AI: Artificial Intelligence, a computer application that can learn, adapt to changing environments based on previous experiences.
  - API: Application Programming Interface, a software intermediary that allows two applications to talk to each other.
  - WAF: Web Application Firewall, an application firewall that applies a set of rules to an HTTP conversation.
  - Agile: A software development strategy based around developing in iterations or sprints.

1 Introduction

1.1 ACKNOWLEDGMENT

Technical advice and guidance were given by Dr. Yong Guan, Dr. Berk Glumezoglu, and Dr. Akhilesh Tyagi. Andrew Dakin provided technical advice and product specifications and a test domain from Cylosoft.

1.2 Problem and Project Statement

Problem Statement: Cylosoft is a company that designs, codes, and hosts websites. These websites are often probed and tested by bots, hackers, and spammers. The web servers at Cylosoft generated text log files as URLs are hit, and Cloudflare acts as a web application firewall (WAF). Cylosoft uses Cloudflare as its firewall, which has both Cloudflare generated rules and customer-generated rules, and there are gaps in the rules that can be improved.

Solution Approach: This problem can be solved by creating a web server-side console application that gathers logs and sends them in real-time to a cloud database. The database will then be queried by a database scanner, which will look for potential bots, hackers, and spammers and generate firewall rules accordingly. The project's output will be real-time WAF rules for the Cylosoft web servers, generated by our scanning application and based on web traffic patterns. A web application will also be created to provide a user-friendly interface for our client's employees to view data in our database.

1.3 Operational Environment

Our project's component is software that will run on hardware (computers, servers, etc.) located at the Cylosoft office. While we do not need to consider conditions such as heat, weather, and moisture for our product, we must ensure that every aspect of our code can run smoothly on Cylosoft's system.

1.4 Requirements

Functional:

- Microsoft ISS will generate text-based log files of each web request.
- A console-based application will be created, which will find the current log file for the site that is being monitored. The application will then send the log file to the web service to be stored and processed.
- The Azure web service will process and store relevant information from IIS log files.
- The database scanner should have access to the processed IIS log data.
- A database scanning application will be created to monitor the log files and identify suspicious web activity using the Cloudflare API.
- When suspicious activity is detected on the server, the potentially dangerous IP will be blocked or for twenty-four hours.
- The system should be able to run in a "log-only" mode for testing on a live site.
- A web application will be created to provide a user-friendly interface for viewing the contents of our database.

Non-Functional:

- The system should be able to perform network inspection for as much uptime as possible.
- Building and scaling the system should be easy to manage.
- The system should be able to accommodate large volumes of traffic.
- Access to the system should be restricted to only approved users.
- The system should block suspicious network traffic upon identifying it.

Environmental:

- Servers and databases should be able to function indoors.
- Rules and IP blocks of network firewalls should be updated per Cylosoft standards.
- Servers should function with web requests from any distance.

Economic:

- All testing should be completed using one domain/server, which will be provided by the client.
- All other design and implementation should be completed using development platforms, IDEs, etc. at no extra cost.
- The project should be completed with six team members, with each team member working three hours per week.
- The project should be designed and implemented by May of 2021.

1.5 Intended Users and Uses

Our product's intended end-user is Cylosoft, with the product operating on Cylosoft's existing software system. Our work is intended to be scalable to meet the needs of Cylosoft as they expand their software in the future.

1.6 Assumptions and Limitations

Assumptions:

- Supplies to accomplish this system will be provided by Cylosoft.
- Team members working on the project will be able to access the resources needed to accomplish the project.
- All equipment utilized will be in working condition.
- The scope of the project will not change for the duration of its life cycle.
- The project source code will be utilized after the project is completed.

Limitations:

- Existing network bandwidth can't be tied up by traffic generated from our project.
- Project planning shall be done by the end of the fall semester.
- The design and implementation of the project shall be done by the end of the spring semester.

1.7 Expected End Product and Deliverables

We will have a complete design document at the end of this project to help stakeholders in further understanding of the design process we used. A console based application that will read log files from Microsoft IIS. These log files will be parsed and fit the Azure web service database template. Each log file will be assigned to a corresponding website to ensure ease of accessibility of necessary information. The AI will be developed that will have access to the processed IIS log information and will help monitor them to identify suspicious activity. When the database scanner detects suspicious activity it will block or challenge its access to the site for twenty-four hours.

Deliverables:

Status Reports-

- We will be providing bi-weekly reports on our progress and report any shifts in our timeline. The team will also provide the date of all meetings held and the discussions held.

User Manual-

- We will provide a user manual at the end of this document, under Appendix I. The manual will explain how to use, test, and modify the software that we have created.

Project Design Document-

- We will provide a detailed plan and description of the process we used along the development of this application. Many things will be outlined in this document, some of which includes: Gantt chart, price cost, and technologies used and relevant to the project.

Azure Web Service-

- We will create an Azure web service that receives Microsoft IIS logs and stores the relevant data from them in a database.

Console Application-

- We will develop a console application to parse through Microsoft IIS log files and return them in a proper format to fit Azure database fields. Log files will also be assigned their corresponding website so that they can be easily tracked

Database Scanner-

- Develop a database scanner that will be able to find "obvious" threats that Cloudflare would normally not catch. These "obvious" threats will be recorded as rules to improve future responses to that type of attack. The database will then remove the threat.

Testing plan-

- We will provide an example of the log files associated with a website as well as the proper formatting of Microsoft IIS log files inside the Azure database. Also, we will provide the necessary instructions and files to

2. Project Plan

2.1 Task Decomposition

Our initial approach to creating the solution was to break apart the problem into three smaller problems: a console application, a cloud web service, and an artificial intelligence algorithm to make Cloudflare rules. The console application will pull Microsoft IIS logs for a site being monitored and then send them to the web service. The web service will transform the records and fit them into our data structure. Our database scanner will then use the web service logs to generate Cloudflare WAF rules to be applied to the monitored site.

2.2 Risks And Risk Management/Mitigation

Main Tasks:

Building console application - Risk Score: 0.6

- This task was given a risk score of 0.6 because there will be a lot of different things that will have to come together for this task to work properly. This application will have to make connections between the Microsoft IIS log files as well as our web service. To add to that, it also has to look at the IIS log files and extract important information and translate that to a condensed log file to send over.

Building web service - Risk Score: 0.5

Building database service - Risk Score: 0.2

Building database scanner - Risk Score: 0.9

Testing/Quality Assurance - Risk Score: 0.8

- Testing and making sure our systems work together properly without failure might prove to be difficult. As testing

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Proposed Milestones:

Research-

Main Task:

- Research all things related to project (Programming language, Frameworks, API's)
    - Subtask:
        - Share information with group and discuss practicality

Environment Setup-

Main Task:

- Install IDE to be used for console application and AI and corresponding frameworks

Console Application -

Main Task:

- Console-based application will find current log file for site being monitored and send the log file to the web service
    - Subtasks:
        - Establish connection between code and IIS log files
        - Establish connection between code and web service
        - Find log file specific to website
        - Parse log file for necessary information
        - Send parsed log file to web service

Web Service

Main Task:

- Azure web service will store relevant information from logs into database
    - Subtask:
        - Establish connection between code and web service
        - Parsed log file will store data in correct fields in database

Database scanner-

Main Task:

- The database scanner will detect attacks and write new rules that do not exist on the Cloudflare API to improve detection accuracy.
    - Subtasks:
        - Scanner will look through cloudflare rules and remove user if conditions met
        - Detected attacks will result in the creation of new rules not existing within Cloudflare

2.4 Project Timeline/Schedule

Note: Project timeline/schedule reflects our plan as of fall 2020. Some aspects of this schedule are now outdated as we have made adjustments to our project, but creating a new schedule after completing our project would be redundant.

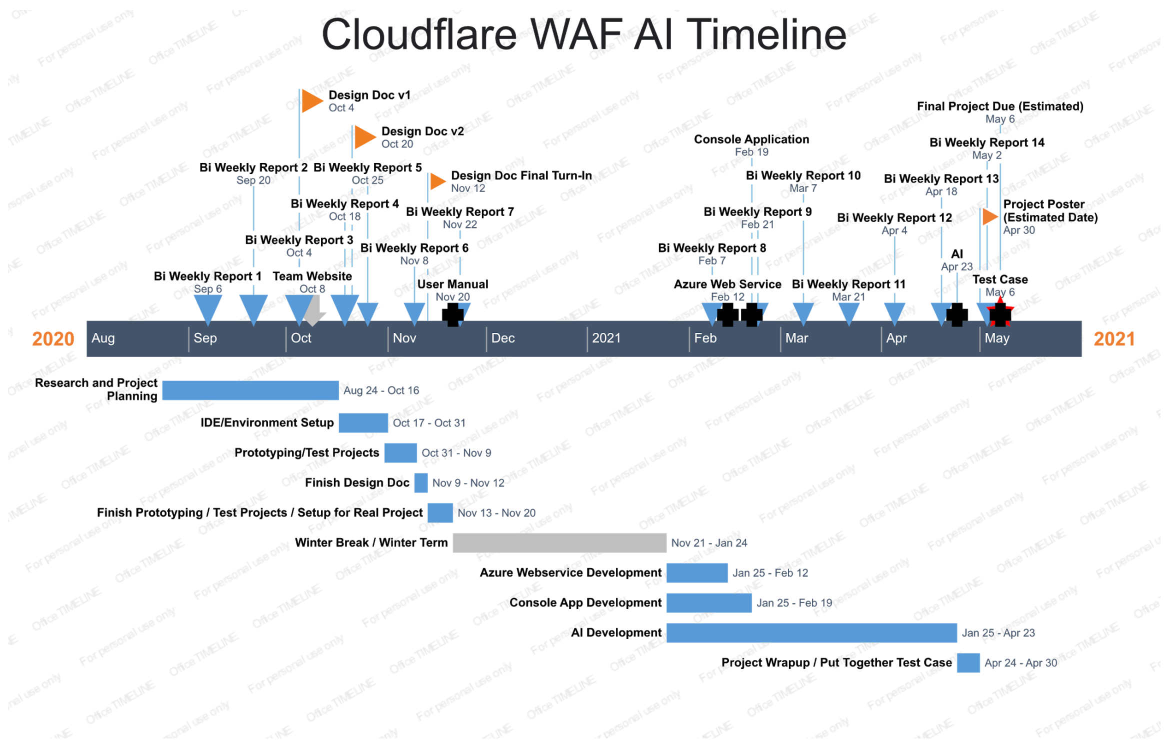


Figure 2.4 Project Schedule

Note on the gantt chart: Listing all subtasks made the above chart unreadable, so the timelines for each subtask are spelled out below.

Note on our development strategy: We plan on using an Agile development strategy with one week sprints instead of the traditional two week sprints. Our project has many small tasks that would be difficult to lump together in two week sprints, and with this semester being remote, we were planning on meeting weekly from the beginning.

Breakdown Into Subtasks

- Research and Planning
  - Information gathered through research was shared with the group each week.
- IDE/Environment Setup
  - The IDE and necessary libraries are to be installed and set up for each developer by October 31
- Prototyping / Test Projects

    - There should be a test project for the web service after the second week, a test console app after the third, and a test AI project by the final week of this task.
- Finish Design Doc
    - The design document should be completed and turned in by November 12
- User Manual
    - There should be a rough user manual (subject to change throughout the project) by November 20th
- Azure web service
    - The first week will be dedicated to setting up a database
    - The second and third weeks will be used to set up automatic filling of the database with test data sent to the webservice.
- Console Application
    - The console application should be able to find log files specific by the end of the first week.
    - By the end of the third week, the console app should be able to send the relevant data to the webservice.
- AI
    - By the end of the second week the AI application should be able to set Cloudflare rules by manual input.
    - The rest of the time for AI will be used to develop the machine learning algorithm
- Project Wrap Up/ Test Case
    - By the project deadline, there should be a presentable test case scenario for the program and our project sponsors should fully understand how to duplicate our setup and take over the project from u s.

2.5 Project Tracking Procedures

With the semester progressing we have utilized different tools to help manage as our project grows. We are using slack to communicate with our client and manage any upcoming issues with our project. For project discussion among our group members, we heavily use discord to manage meetings and share resources that pertain to our project. It gives us the opportunity to brainstorm and finalize ideas for our project. We have also set up a Github repository and GitLab manager so it's ready once we start development in our project. These repositories will help manage changes in our project among all of our members. We will all be able to make changes once our project starts without interfering with anyone's work. Our team also discussed the possibility of using Trello to help keep track of achievements and major milestones of our project. From experience, we can all say trello is a great tool for development management. Aside from these useful management tools, we can all say our team has utilized these to the fullest. We hope to come across any more useful tools as we start development with our project!

2.6 Personnel Effort Requirements

TASK	ORDER	PERSON-HOURS
Research	1st	108
Environment Setup	2nd	18
Console Application	3rd	72
Web Service	4th	36
Database Scanner	5th	126
Web Application	6th	80

Table 2.1 Personnel Effort Requirements

Each of the tasks will initially be taken on by all team members. As the tasks get more in-depth, team members will choose a specific task to focus on. The tasks are ordered because some tasks depend on the previous implementation. For example, the way that the database is formatted depends on how the console application formats the IIS logs. The person-hours are estimates and will depend on how many obstacles our team runs into as we learn how to implement the tasks for the project.

2.7 Other Resource Requirements

- Virtual environment for testing, provided by Cylosoft
- Microsoft Visual Studio, provided by Iowa State University

3 Design

Note: Some aspects of the design section represent our research before we modified our design plan to include a database scanner instead of an AI application. We left some information regarding our plans for the AI and also will show the difference between our new and original design plans.

3.1 Previous Work And Literature

There are many distinguished web security companies out there that are really advanced. These companies include: Crowdstrike, Gigamon, Fortinet, and many more (Admin). Crowdstrike has been used for many high profile cases including the controversy surrounding the 2016 Russian investigation (Admin). Crowdstrike makes use of AI to detect threats in a similar fashion to our project (“About…”). Although we may not be able to compete with as much funding and resources as Crowdstrike we can definitely do our best to create an AI that is efficient and self-sufficient through the use of machine learning.

There is a three part process to our project. The first part involves grabbing Microsoft IIS files and parsing through them to get necessary information; all of this is done using the console application. The second part involves sending data to the web service Azure and making sure that the data is processed and stored correctly. Lastly, the third and final part involves creating an AI that has machine learning capabilities in order to detect attacks based off of the Cloudflare API and adding its own rules as it detects new types of attacks.

In order to grab log files from IIS programmatically we'd have to use the ILogScripting COM Interface (Archiveddocs). As mentioned on the website by Microsoft

"If you want to create a custom component that implements the ILogScripting interface, you need to create an automation (IDispatch) object implementing the methods of the ILogScripting interface.

Your interface should be designed and implemented to handle only one log file query session per module instance. Therefore, each call to ILogScripting::SetInputLogFile and ILogScripting::SetInputServerInstance should reset the record read state"(Archiveddocs).

The second part is relatively straightforward, we'd just check the database to ensure that the IIS log files were properly parsed and formatted to fit, we could query this by using SQL.

Lastly, the AI we create will be the most difficult part of the project and will take the majority of our time. We will be using machine learning and Tensorflow library to assist us. There are many different libraries we could have used like: Keras Python, Theano Python, Scikit-learn python and many more ("Top 8 Pytho..."). However, we decided to go with Tensorflow because it seemed to have the most resources available. Also, tensorflow syntax is considered simpler than the others, when combining those two factors and the power of tensorflow the decision was easy. The best resource we've found has been the tensorflow website, they have tutorials and guides for anyone ranging from the beginner level to advanced ("Machine Learning...").

3.2 Design Thinking

We as a group, based our design thinking in the perspective of the client. The project was described into three major parts that build on top of each other. The group altogether conducted research to find the best resources and programming languages to ensure a high quality product. We have been deciding based on what is most compatible and efficient to help in the progression of the project and the perspective of the client. Our console app takes in IIS files and will require it to be written in a certain programming language. Our database scanner and web application will be programmed to interact efficiently with our database. Primarily each part of the project design is associated with specific resource and language requirements, so group design choices were focused on compatibility and efficiency as a whole.

3.3 Proposed Design

As mentioned in previous sections, the work we need to complete can be broken into a couple of components.

- Web Server
    - This is the web server that contains the website our project is in charge of protecting. Our project should be able to work for a variety of different web servers, so I will not spell out the details here.
- Console Application
    - This application will be responsible for finding, parsing, and sending relevant information from IIS log files to the Azure web service. We have a prototype version, written in Python, that can parse IIS logs and extract the relevant data. It does not yet find log files or send relevant information. This application will make use of REST protocols for sending data.
- Azure web service
    - This web service will consist of a database for storing IIS log information relevant to the AI. This will make use of REST protocols for transferring data between the database, the console application, and the AI. Since we are using Azure, the database will be scalable should the number of web servers under our protection increase.
- Database scanner
    - The database scanner will be responsible for analyzing the relevant data from the database and creating new Cloudflare rules based on that data. It will also make use of REST protocols in receiving data and updating the Cloudflare rules.
- Cloudflare WAF
    - This is the firewall which will block unwanted traffic from reaching the web server. It's rules will be updated in real time by our AI through its API. Administrators will also be able to set rules manually in case of an incorrect assumption by the AI or an update to Cylosoft standards.
- Web application
    - The web application will be responsible for providing a user friendly interface for our client to easily view and filter the contents of our database.

3.4 Technology Considerations

-Visual Studio IDE

- Strengths-
    - Created by microsoft and is compatible with Azure, SQL, Python
    - Easily manage frameworks
    - Most members of our team are familiar with the UI of Visual Studio.
- Weaknesses-
    - Can be slow at times
    - Uses a large amount of memory that results in slowing down your computer.

Alternatives to Visual Studio include: IntelliJ, Notepad++, Eclipse, and many more. We decided to go with Visual Studio because it comes from microsoft and made to be compatible with microsoft services such as IIS.

-Python Language

- Strengths-
  - Python is capable of interacting with most other languages thanks to the Python Package Index.
  - Has a large standard library which results in reduction of code length
  - Python is open source
  - Python is Object Oriented and has its own unit testing framework.
- Weaknesses-
  - Python executes with help from an interpreter which causes it to slow down.
  - Has design restrictions that are often reported by other Python developers.

Alternatives to this language are C, Java, C++, and many other languages. However, we chose to work with python because it is a growing language and has a lot of documentation as well as framework support. Also, after researching AI we came to the realization that most of the material was done in Python.

-TensorFlow (AI Library) (Research from fall and spring semester)

- Strengths-
  - High Performance
  - Large Community Support
  - Highly Parallel, can pipeline easily
  - Great Graph visualizations
  - Frequent updates, backed by Google
- Weaknesses-
  - Benchmark tests are low
  - Very low level with high learning curve
  - Unique structure resulting in potential difficulty while debugging
  - No windows support
    - Can be installed on windows through use of Python package library

Alternatives to TensorFlow include: Keras Python, Theano Python, Scikit-learn python and many more. We decided to go with TensorFlow because of its high performance and great visual tools. There is also a lot more information online for us to reference and learn from.

3.5 Design Analysis

From a design perspective, our plan has been effective. We have made fairly major changes to our design plan as the spring semester has progressed, mostly due to new ideas and requirements as we have interacted with our client. These changes include using a static database scanner instead of an artificial intelligence, changing the coding language of our console application from Python to C#, and adding a web application. These changes are detailed more in Appendix II.

3.6 Development Process

We as a team are following the *agile* development process for this project. With this project primarily being software based, we unanimously agreed this was the appropriate choice. In the beginning stages of our development, we had already implemented continuous planning and collaboration with our project. We have produced realistic timelines and team expectations to produce a high quality finished product. With each of us bringing different perspectives to this end goal, we are continuously learning and working together to make sure everything is in order. Once we hit the prototype stage, our group will be able to gather feedback and adapt to the necessary changes that may occur in this development process.

3.7 Design Plan

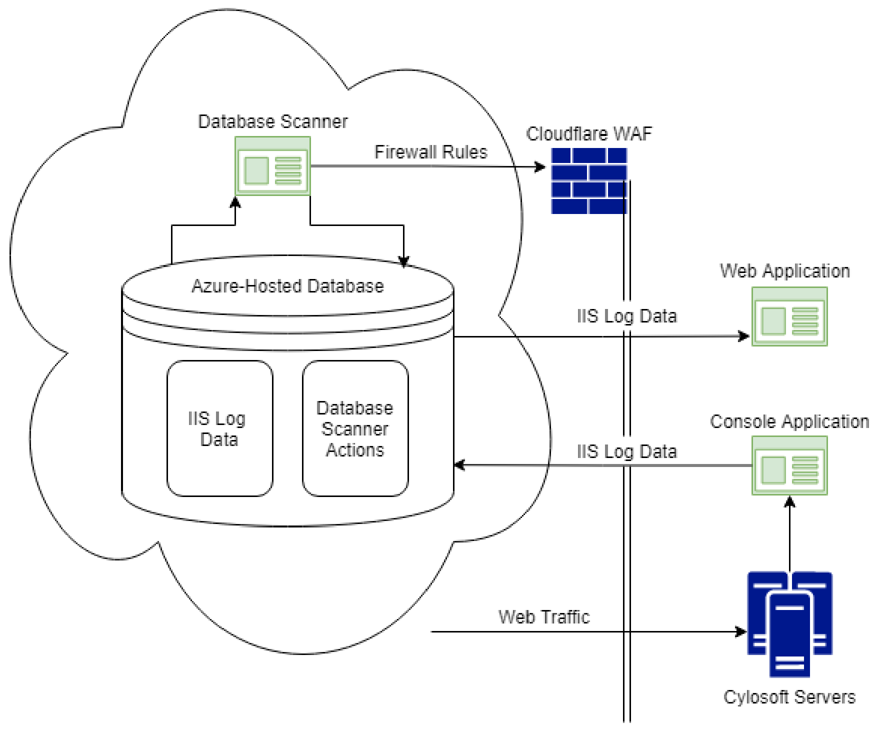


Figure 3.1 Design Plan

(For a full list of requirements, see section 1.4)

- Meeting Functional Requirements:
    - The web server and console application modules will meet the functional requirements of generating log files for web requests and collecting log files.
    - The Azure web service database module will meet the functional requirement of storing and processing log files.
    - The database scanner application will meet the functional requirements of monitoring and updating usage rules, and denying access to suspicious users.
    - The web application will fulfil the functional requirement of providing a user-friendly interface for our client's employees.
- Meeting Non-Functional Requirements:
    - The Azure web service module will be active at all times to keep the system running for as much uptime as possible.
    - The database and console application modules will not have limits on the number of logs that they can process to ensure that the system will be capable of handling large volumes of traffic and be easy to scale.
    - Security checkpoints will be implemented on the database and console application modules to ensure that it can only be accessed by approved personnel.
    - The Cloudflare module will fulfill the non-functional requirement of blocking suspicious network traffic.
    - The web application will have an authentication system that will fulfil the non-functional requirement of only allowing authorized personnel to view the contents of our database.

4 Testing

Testing for our project will be critical. To accomplish this, testing will be broken down into the following types of tests: Unit Testing, Interface Testing, and Acceptance Testing.

Each type of testing will follow roughly the same procedure. Our team will come up with what needs to be tested. Upon agreeing on what requires testing, tests should be written as well to help determine if what is tested performs as intended. When the team is done writing tests, the next step is to carry out the tests on the intended units, interfaces, and systems involved. After the tests come back, the team will meet to discuss what went as expected and what didn't. From this meeting we can come to a decision on next steps to perform to get everything working properly. Keeping accurate documentation on this whole process will also be vital in making sure everyone is on the same page about testing that has happened and directions to take proceeding the meeting.

4.1 Isolation Testing -Individual component testing

- Console Application
    - Used visual studio debugger to validate variable values.
    - Used sample log files provided by our client.

    - Used command line printing to validate variable behavior.
    - Ex: to test the log file parsing method, I would input sample log file and debug output variable. I would check values in different rows and columns of the output variable (focusing on edge cases), and ensure that they matched correctly with entries in the sample log file.
    - Ex: to test the date/time method, I would create a variable representing the current date, input it to the date-time method, and ensure that the correct conditional route was taken.
    - Ex: to test the database insert method, I would create a string with sample values in SQL format and input it to my insert method. I would run the method, then check the contents of the database to ensure that data could successfully be inserted.
- Web Application
    - Used visual studio debugger to validate variable values.
    - Used chrome debugging console to identify dependency errors.
    - Used command line printing to validate variable behavior.
    - Ex: to test the database query method in the web application, I would redirect the output of the method to the console. I would then run the method and ensure that the data outputted to the console matched the data in the database.
- Database Scanner:
    - Used visual studio debugger to validate variable values.
    - Used Log files with known attacks provided by our client.
    - Ex. to test the flooding attack method we would use an log file containing a known flooding attack as an input. We would then ensure that our method outputs the IP address of the attacker.

4.2 Interface Testing -Custom testing applications

- Console Application:
    - Set up IIS on my local machine to replicate the exact environment (file structure, etc.)
    - Created a test log data table in the database.
        - Used the exact specifications of the production database table.
        - Created in order to test new versions of the application that needed to be tested after real data was in the production database.
    - Created a Python script to simulate web traffic.
        - Used to simulate live additions to a log file.
        - Script took lines from a sample log file and inserted them into the local IIS log files at the rate of two lines per second.
        - A second simulator script was also created to ensure that the code could handle multiple live sites.
    - Ensured application robustness by exposing code to edge cases and uncommon situations.

        - Made sure that the application could handle sites being added and deleted in the middle of a scanning cycle.
        - Made sure that the application could resume scanning where it left off if the app was abruptly stopped (by storing and retrieving values from the database).
- Web Application:
    - Most web application testing was done either by individually testing components (discussed above) or testing on our client's test server (discussed below).
    - Set up the Visual Studio .NET solution in the local IIS web folder to ensure that solution could run properly.
- Database Scanner:
    - Pulled data from the existing database that was gathered through the console application.
    - Ran through local logs to ensure the scanner was getting controlled data.

4.3 Acceptance Testing -Client test server

- Console Application:
    - Our client ran our console application code low risk/low volume server at his company.
    - The client test server ran ten different sites and averaged about 60,000 entries per day.
    - The client test server helped us catch errors that we were not able to think of or generate ourselves (discussed below in section 4.4).

- Web Application:
    - Connected the web application to the live database from our console application acceptance testing.
    - Ensured that the testing environment would be identical to the production environment.
    - Transferring the web application from being hosted locally to hosted on the client test server ensured that all dependencies were connected to the application and could be transferred to production without errors.
- Database Scanner:
    - Connected the database scanner to the live database from the console application.
    - Ensured that the testing environment would be identical to the production environment.
    - The test ran through all logs that would be received from the console application.

4.4 Results

Using the three steps above in chronological order for testing our applications was very beneficial to our project quality. For the first step, isolation testing, the goal was to make sure each individual component was functioning properly. This resulted in us catching errors like the wrong time zone returned by C#'s datetime library, and our IIS folder parser mistaking files for folders and trying to open them. The second step, interface testing, ensured that each of the individual components tested in the first step could work smoothly together. This step is where we implemented try-catch loops to handle errors across the application, and where we generally ensured that the code could run in an ideal environment. For the final step, acceptance testing, we utilized our clients test server to catch unusual errors that we could not foresee in our previous testing steps. This helped us catch multiple errors in the console application, such as extremely long URL paths and emojis in URL queries, and also helped us make sure that the web application could efficiently query the database with over a million entries. Using these three testing steps, we were able to be confident in the robustness and reliability of our project.

5 Implementation

- Azure web service
  - Set up Microsoft SQL Server Management Studio (SSMS) to have an interface to easily work with our database.
  - Obtained connection string to database and credentials to sign into (SSMS).
  - Initialized database tables:
    - Log data table
    - Test log data table
    - Server index table
    - Site index table
    - Error logging table
- Console Application
  - Set up IDE in Jupyter Notebook with Python
  - Created a fully functioning application in Python (Python version not used, so more details added below for .NET version).
  - Set up IDE in Visual Studio with C# in a .NET console application solution.
  - Implemented initial database connection function.
  - Implemented error catcher that logs all errors in the database (other than database connection errors).
  - Implemented method to retrieve all log files from all sites on the server that the app is running on.
  - Implemented date and time check to reset the count values at a new day (the app runs on UTC time, as requested by our client).
  - Implemented main loop that inserts new web traffic from each site into the database. Loop repeats once per minute as requested by our client.
- Database Scanner
  - Admin attack scanner searches for attempts to reach or login to the admin page of a webpage. It checks to see if a uri contains specific words such as "admin" or "login"
  - Flooding Attack Scanner searches for flooding based attacks and adds the IP address to a list of high-risk users.
  - OS Scanner seeks out any Linux user-agent IP's and adds them to a list of high risk users.

    - SQL Attack Scanner searches for specific malicious keywords that are entered by the "user" and adds the IP address to a list of high-risk users.
- Web Application
    - Added a web application to the console application .NET solution.
    - Imported the GUI framework within our .NET solution.
    - Attached to required dependencies in order to operate the GUI framework in .NET.
    - Set up an authentication system, complete with a separate login page.
    - Connected the web application code to our Azure database via a connection string in the configuration file.
    - Implemented paging, so that large amounts of data could be easily viewed.
    - Implemented filtering, so that only desired information can be displayed.

6 Closing Material

6.1 Conclusion

As we conclude our senior design project, we are pleased with the work we have done and the product we were able to deliver given the difficult circumstances that we were operating in this semester. We learned many lessons in adaptation while implementing this project. Most importantly, we worked closely with our client through this spring semester and made major changes/additions to our project based on his feedback (detailed in Appendix II). The timing of these changes resulted in a greatly reduced time frame for completing our deliverables, but we were able to communicate with our advisor and ensure that we completed as much of the deliverables as we could, and that the most useful aspects of the project were completed first. This ensured that our client had the maximum amount of useful deliverables by the end of the semester.

6.2 References

About CrowdStrike: Cloud-Native Endpoint Protection. 5 Aug. 2020, www.crowdstrike.com/about-crowdstrike/.

Admin. Admin. 22 July 2020, www.thesoftwarereport.com/the-top-25-cybersecurity-companies-of-2019/.

Archiveddocs. "IIS Logging Overview." Microsoft Docs, docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms525410(v=vs.90).

"Machine Learning Education : TensorFlow." TensorFlow, www.tensorflow.org/resources/learn-ml.

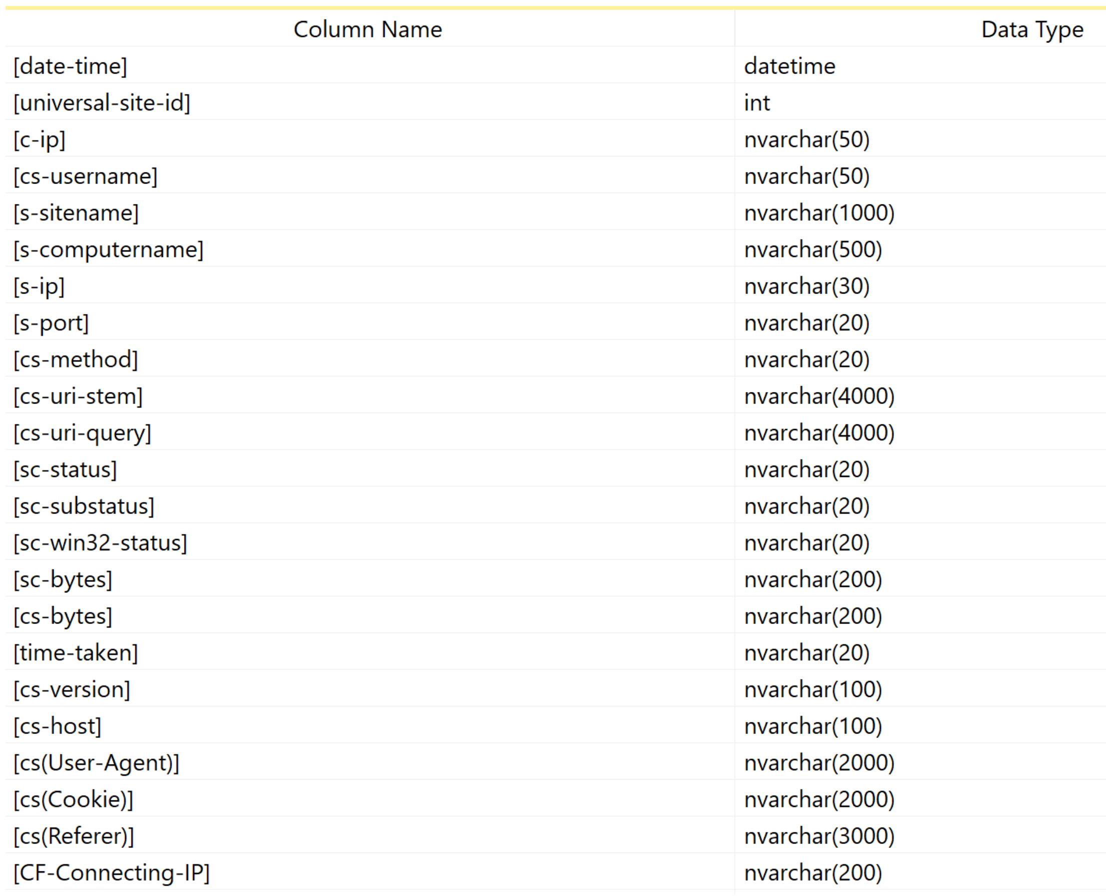
“Top 8 Python Libraries for Machine Learning & Artificial Intelligence.” Hacker Noon, 20 Oct. 2020, hackernoon.com/top-8-python-libraries-for-machine-learning-and-artificial-intelligence-yo8id3031.

7 Appendices

7.1 Appendix I -User Manual

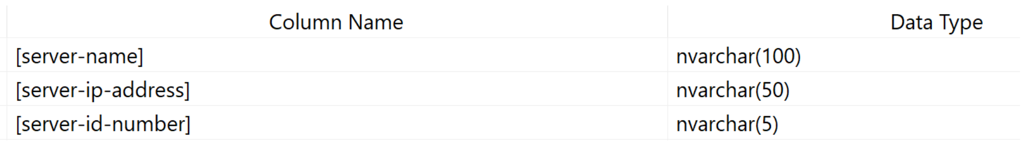
The following user manual is divided into three main sections: the console application, the web application, and the database scanner.

- Console Application:
  - Step 1: Download the source code from the project git repository: https://git.ece.iastate.edu/sd/sdmay21-16
  - Step 2: A SQL database must be initialized so that the console application has a place to send the IIS log Data. The database must have to following tables with the following values:
    - LogTable and TestLogTable with the following column specifications:



Column Name	Data Type
[date-time]	datetime
[universal-site-id]	int
[c-ip]	nvarchar(50)
[cs-username]	nvarchar(50)
[s-sitename]	nvarchar(1000)
[s-computername]	nvarchar(500)
[s-ip]	nvarchar(30)
[s-port]	nvarchar(20)
[cs-method]	nvarchar(20)
[cs-uri-stem]	nvarchar(4000)
[cs-uri-query]	nvarchar(4000)
[sc-status]	nvarchar(20)
[sc-substatus]	nvarchar(20)
[sc-win32-status]	nvarchar(20)
[sc-bytes]	nvarchar(200)
[cs-bytes]	nvarchar(200)
[time-taken]	nvarchar(20)
[cs-version]	nvarchar(100)
[cs-host]	nvarchar(100)
[cs(User-Agent)]	nvarchar(2000)
[cs(Cookie)]	nvarchar(2000)
[cs(Referer)]	nvarchar(3000)
[CF-Connecting-IP]	nvarchar(200)

- ServerList with the following column specifications:



Column Name	Data Type
[server-name]	nvarchar(100)
[server-ip-address]	nvarchar(50)
[server-id-number]	nvarchar(5)

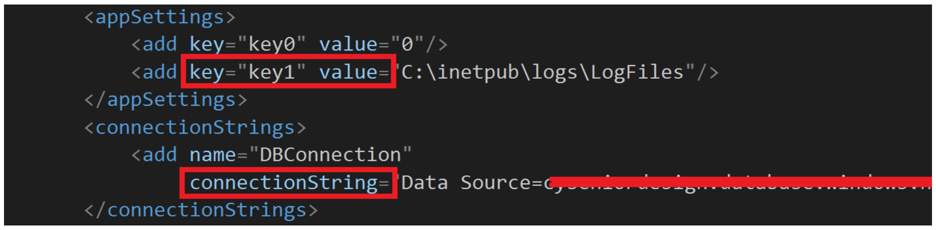
- ServerValues with the following column specifications:

Column Name	Data Type
[server-id]	nvarchar(10)
[site-id]	nvarchar(10)
[site-name]	nvarchar(500)
[site-url]	nvarchar(500)
[universal-site-id]	int
[folder-path]	nvarchar(200)
count	nvarchar(20)
software	nvarchar(200)
version	nvarchar(100)
date	nvarchar(100)
fields	nvarchar(2000)

- ErrorLogging, with the following column specifications:

Column Name	Data Type
[utc-time]	nvarchar(100)
fatal	nvarchar(10)
[error-message]	nvarchar(3000)

- Step 3: Obtain the connection string of your database and add it to the 'connectionstrings' section of the app.config file.
- Step 4: Obtain the absolute path to your root IIS log directory and add it to the 'values' section of the app.config file.



```
<appSettings>
    <add key="key0" value="0"/>
    <add key="key1" value="C:\inetpub\logs\LogFiles"/>
</appSettings>
<connectionStrings>
    <add name="DBConnection"
    connectionString="Data Source=[redacted]
</connectionStrings>
```

Figure AI.1 Console Application Configuration

- Step 5: Open the root .NET solution for the console application code in Visual Studio as an administrator and run it by clicking the green play button at the top of the screen.

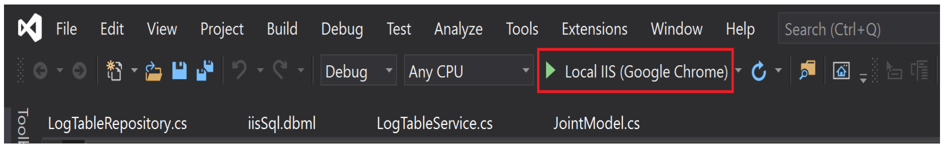
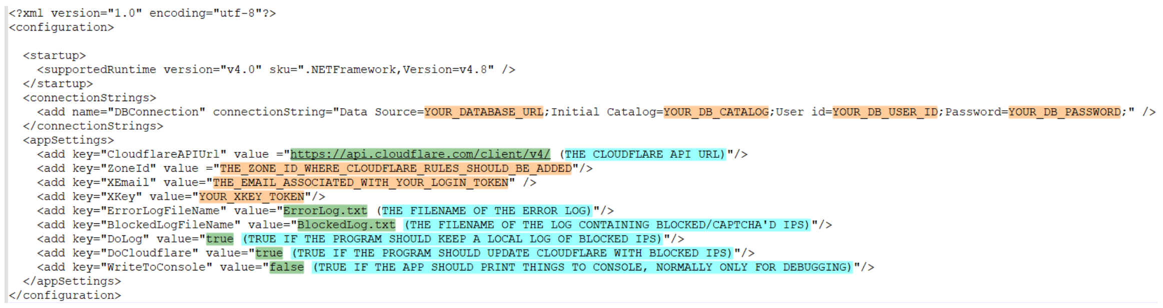


Figure AI.2 Run Console Application

    - Step 6: The console application is now running! Check the terminal in Visual Studio to make sure that the code did not throw an immediate database connection error. If it did not, then the code successfully ran and any future errors will be logged in the ErrorLogging table of your database.
    - Note: For details on how to test the console application, please refer to section 4 of this document.
- Web Application:
    - Step 1: Download the source code from the project git repository: https://git.ece.iastate.edu/sd/sdmay21-16
    - Step 2: Ensure that you have an SQL database initialized and properly configured (see step 2 of Console Application above for details).
    - Step 3: Obtain the connection string of your database and add it to the 'connectionstrings' sections of the web.config file.
    - Step 4: Obtain the login credentials from the web.config file, and change them if you would like.
    - Step 5: Open the root .NET solution for the web application code in Visual Studio as an administrator and run it by clicking the green play button at the top of the screen.
    - Step 6: Navigate to the web address that you hosted the web application on in step 5.
    - Step 7: You should now see the login screen of the web application and be able to log in using the credentials obtained in step 4. After successfully logging in, you will be able to see and filter the database data using the web application.
    - Note: For details on how to test the console application, please refer to section 4 of this document.
- Database Scanner:
    - Step 1: Download the source code from the project git repository: https://git.ece.iastate.edu/sd/sdmay21-16
    - Step 2: Ensure that the console application and database have been properly initialized as specified in earlier steps.
    - Step 3: Add the following information to the app.config file:
        - Orange is necessary information you need to add
        - Green is information you have the option to change if you wish
            - Blue explains what changing the green information does



```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
  <connectionStrings>
    <add name="DBConnection" connectionString="Data Source=YOUR_DATABASE_URL;Initial Catalog=YOUR_DB_CATALOG;User id=YOUR_DB_USER_ID;Password=YOUR_DB_PASSWORD;" />
  </connectionStrings>
  <appSettings>
    <add key="CloudflareAPIUrl" value ="https://api.cloudflare.com/client/v4/ (THE CLOUDFLARE API URL)"/>
    <add key="ZoneId" value ="THE_ZONE_ID_WHERE_CLOUDFLARE_RULES_SHOULD_BE_ADDED"/>
    <add key="XEmail" value="THE_EMAIL_ASSOCIATED_WITH_YOUR_LOGIN_TOKEN" />
    <add key="XKey" value="YOUR_XKEY_TOKEN"/>
    <add key="ErrorLogFileName" value="ErrorLog.txt (THE FILENAME OF THE ERROR LOG)"/>
    <add key="BlockedLogFileName" value="BlockedLog.txt (THE FILENAME OF THE LOG CONTAINING BLOCKED/CAPTCHA'D IPS)"/>
    <add key="DoLog" value="true (TRUE IF THE PROGRAM SHOULD KEEP A LOCAL LOG OF BLOCKED IPS)"/>
    <add key="DoCloudflare" value="true (TRUE IF THE PROGRAM SHOULD UPDATE CLOUDFLARE WITH BLOCKED IPS)"/>
    <add key="WriteToConsole" value="false (TRUE IF THE APP SHOULD PRINT THINGS TO CONSOLE, NORMALLY ONLY FOR DEBUGGING)"/>
  </appSettings>
</configuration>
```

Figure AI.3 Database Scanner Configuration

- Step 4: Open the root .NET solution for the console application code in Visual Studio as an administrator and run it by clicking the green play button at the top of the screen.

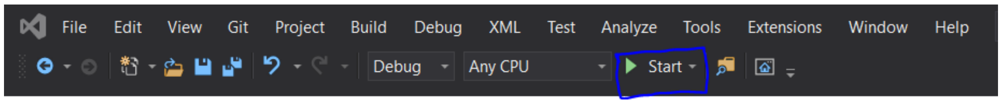


Figure AI.4 Run Database Scanner

- Step 5: The database scanner is now running! Depending on the configuration you chose, check your blocked log file and/or access rules on Cloudflare to see if any IPs have been flagged.
- Note: For further details on how to test the database scanner, please refer to section 4 of this document.

7.2 Appendix II -Design Changes

While we were implementing our project, we made several major changes/additions, based on requests and feedback that we received from our client. We will discuss three of those changes in this section: the switch of the console application environment, the switch away from using artificial intelligence, and the addition of a web application. From a high-level perspective, here is our old conceptual sketch first, and then our current conceptual sketch below it. You can see some of the changes we made reflected in the differences between the two sketches, such as the addition of the web application and the change from an AI to a database scanner.

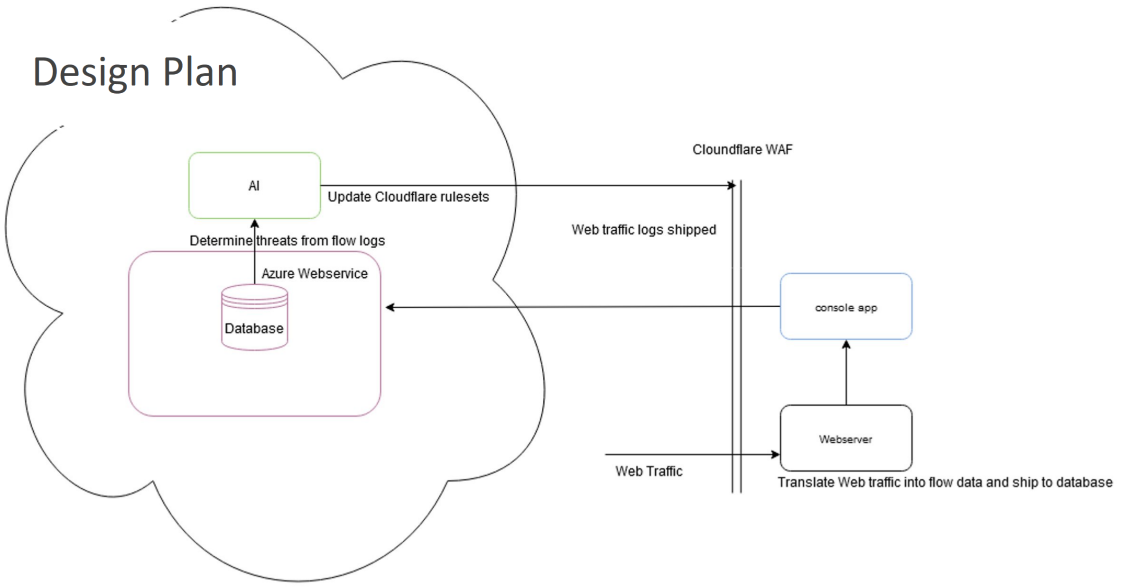

Design Plan
AI
Update Cloudflare rulesets
Determine threats from flow logs
Azure Webservice
Database
Cloundflare WAF
Web traffic logs shipped
console app
Webserver
Web Traffic
Translate Web traffic into flow data and ship to database

Figure AII.1 Old Conceptual Sketch

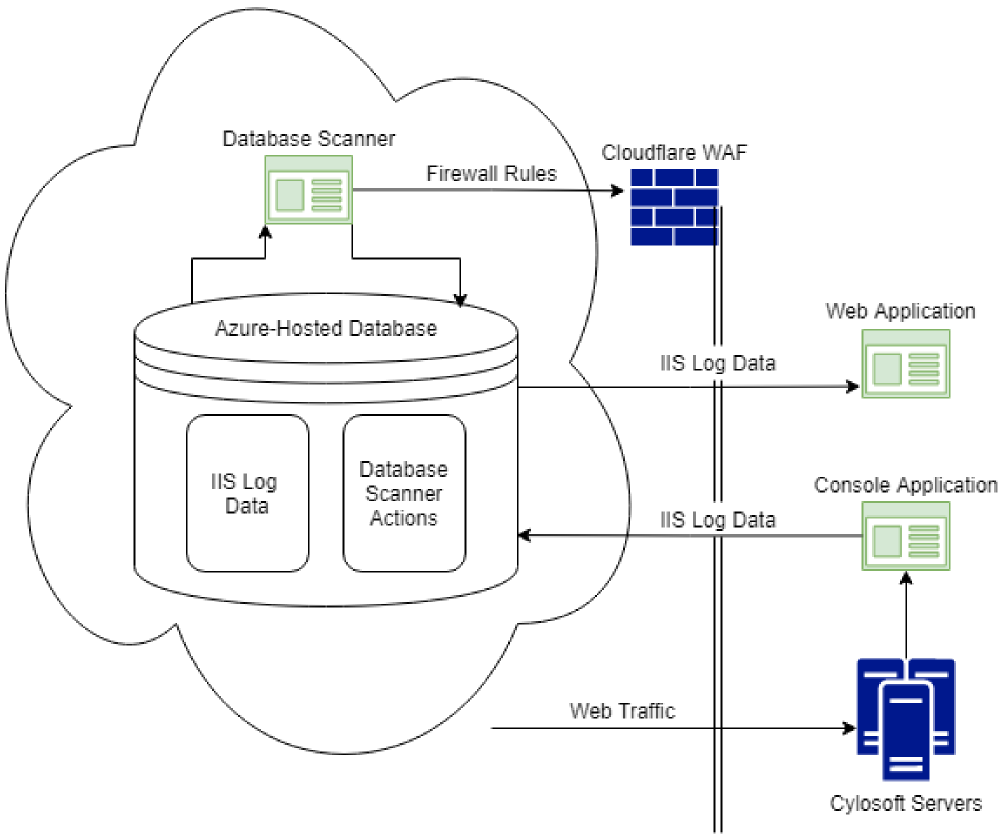


Figure AII.2 Current Conceptual Sketch

- Switching Console Application Environment
    - The console application was one of the first aspects of our project that we implemented. Originally we asked our client if he had a preference for which language to code the console application, and he said to choose whatever would work best for us. We chose Python because of its vast and useful libraries for working with files since our console app works with the IIS system. After we completed the app in Python, our client realized that functionality of the console app is very useful and wanted to integrate it with other applications at his company, such as the web application that we created (discussed later).

      For this reason, he requested that we rewrite the console application code in C# using the .NET framework, since that is the standard for applications on his servers. Making the transition between coding languages included using different methods for parsing files, different types of configuration files, and different libraries for working with SQL. This taught us how to find solutions from multiple angles, and how to quickly learn new skills in order to satisfy our customer's requests.

- Switching to Static Database Scanner Instead of AI
    - Our original project description from our client included an artificial intelligence application that would train itself on the data in our database. It was a good idea in theory, but as we began to work on implementation, we decided with our client that an AI application would not be feasible. Our client did not have much experience in AI, so when we were figuring out how to train our machine learning model, we realized that our client did not have enough labeled training data to train the model. We also looked at finding open sourced training sets online, but determined with our client that we could not find a set that would be specific enough for our project.

      So instead of an AI application, we decided to move forward with a static scanning application that would use methods to query the database and make determinations based on preset thresholds. Because we were so late in our project implementation when we made this decision, we did not have much time to implement the static scanner, but we made sure to start with the core framework so that it would be easily scalable and therefore more beneficial for our client.

- Adding Web Application
    - Half way through this spring semester, our client realized that it would be beneficial to add a web application to our project. This is because he did not previously have a way of organizing the data in his IIS log files, so if he wanted to view network traffic, he would have to look at each log file on each site of each server at his company. Our console application simplified this by organizing all of that network traffic into one central database. Our client realized this was very convenient for him and began running SQL queries on our database instead of viewing individual IIS logs to save time.

      This led to him asking for us to create an official .NET web application using a GUI framework that the company owned. This required us to quickly learn how to implement a user-friendly interface using their GUI, which we had never used before. We also had to modify the way the UI queried the database to deal with the large amount of data that our console application had collected, and to ensure that the code was scalable in case our client wanted to expand its reach.