HIGHLIGHT = WORK ON
CPRE 492 Bi-Weekly Report #6

Report Period: March 29th - April 12th

Group Number: 16

Project Title: Cloudflare WAF AI

Client/Advisor: Cylosoft; Dr. Yong Guan, Dr. Berk Gulmezoglu

Team Members:

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

Summary:

During this period, now that we have finalized sticking to a static database scanner, we have been implementing small amounts of attack detection based on what we are trying to account for within our massive database. Last report period, we were able to implement a rough draft for each of our various attack detections. Now we are testing each detection to our client's standards to make sure each detection is pulling the right IP addresses, and creating the correct rules and block time duration for Cloudflare's WAF. The final stages of our web application are being implemented and tested. We want to make sure we do not run into errors based on any user requests, whether it be server or database errors that may occur during the start-up development of our application.

Past weekly Accomplishments:

During the reporting period, each member has been assigned to implement their desired method of attack detection. Ryan was assigned to work on his Flood Attacking detection for the database scanner. Last period, he was able to get a draft code implementation for returning IP addresses. We are able to say the same for the rest of the Scanning Team of our Project. Giovanni took the attack method of SQL injection to implement for the database scanner. As he was implementing he felt he did not account for EVERY single type of SQL injection attack. That leaves room for discussion on what the client has seen as very common and what should be implemented for that method of detection. Jordan chose the Admin attack method to implement. In this case, you can be straightforward with this method of detection. These attacks are brought to the attention only when a user is maliciously entering the database with the credentials of an admin user. Preston chose the OS attack method. With OS detection, he was able to get the IPs of consoles based on what OS was being run at the time. As stated in the summary, each method of detection is operational to a certain extent. With this upcoming period, we anticipate having a fully operational static scanner for all various attacks. Eric took the liberty of merging

Cloudflare's API scanning tools. This helps us determine if our database scanner is aligned with Cloudflare's set rules and the correct duration of the IP address being blocked. Our Client did not want an IP address to be blocked permanently (on account of the scanner mistakenly flagging something as malicious) rather it be blocked for a couple of hours to run further testing to see if the block was correctly applied or needs modification. Finally, Ben is working on the final touches for the web application. There have been some connectivity issues but all resolved on account of correct configuration within desired servers with the client.

Pending Issues:
Currently no pending issues.

Individual Contributions:

Team Member:	***Individual Contributions***	***Hours this week***	***Hours Cumulative***
Ryan Burgett	Worked further on the Flooding attack detection implementation. Two more different types of Flooding attacks have been added to be accounted for.	7 hours	41 hours
Giovanni Mejia	Worked further on the SQL injection detection. Some things were previously unaccounted for and were added for the client's needs.	7 hours	40 hours
Jordan Heim	Worked further with the Admin attack Detection. Previously there was an issue with the CSUser_agent in the static code. Things have been modified for more efficient connectivity and no errors.	7 hours	37 hours
Eric Reuss	Merged together Cloudflare tools for the static database. The reasoning is for an efficient environment	7 hours	40 hours

	between the database and Cloudflare's API. Now that everything is merged and connected together, it will help the database team with testing and making sure correct IP blocks are placed.		
Presiian Iskrenov	Worked further on the OS console detection. The first draft implemented for Linux-based consoles, we need a broader range as there are Operating Systems to account for.	7 hours	39 hours
Benjamin Fedderson	Worked further with the filters functionally of the web application. Previously had connectivity challenges that were resolved with the client.	7 hours	44 hours

Plans for the upcoming week:

Our hope is to have a fully functional database scanner as the semester is coming close to an end. Each member of the database scanning team needs to finalize their detection attack methods and make sure each IP list is correctly being gathered. Now that we have a full merge with our Cloudflare API, we can also test to see if the correct blocks are being made and that they are being blocked for the correct amount of time for our client's desire. We would like the Console Application and Web application to be fully functional by the next reporting period. That requires some small touches. Then the database will be fully functional by that time as well.

Screenshots from the work this reporting period:

Here are some things that were merged with the Cloudflare scanning tools.

The screenshot shows the implementation of the created firewall rules and syncing with our database on Azure. It takes in the JSON string data and returns it as the URI of the originally created resource. All of these functions act together to create the firewall rules and update accordingly based on the Database scan and the associated firewall rules.

```
public async Task<Uri> CreateFirewallRuleAsync(FirewallRule rule)
{
    //Tell the json converter to ignore null variables
    JsonSerializerSettings settings = new JsonSerializerSettings();
    settings.NullValueHandling = NullValueHandling.Ignore;

    //Serialize the firewall rule object
    string jsonString = "[" + JsonConvert.SerializeObject(rule, settings) + "]";
    StringContent sc = new StringContent(jsonString, Encoding.UTF8, "application/json");

    //Have to set this header manually for some reason
    sc.Headers.ContentLength = jsonString.Length;

    Console.WriteLine(await sc.ReadAsStringAsync());
    //Send the request
    HttpResponseMessage response = await client.PostAsync(ConfigurationManager.AppSettings["CloudflareAPIUrl"]
        + "zones/" + ConfigurationManager.AppSettings["ZoneId"] + "/firewall/rules", sc);

    // return URI of the created resource
    return response.Headers.Location;
}

public async Task<Uri> DeleteFirewallRuleAsync(string id)
{
    HttpResponseMessage response = await client.DeleteAsync(ConfigurationManager.AppSettings["CloudflareAPIUrl"] +
        "zones/" + ConfigurationManager.AppSettings["ZoneId"] + "/firewall/rules?id=" + id);
    return response.Headers.Location;
}

//Make sure the id is set
public async Task<Uri> UpdateFirewallRuleAsync(FirewallRule rule)
{   //Tell the json converter to ignore null variables
    JsonSerializerSettings settings = new JsonSerializerSettings();
    settings.NullValueHandling = NullValueHandling.Ignore;

    //Serialize the firewall rule object
    string jsonString = "[" + JsonConvert.SerializeObject(rule, settings) + "]";
    StringContent sc = new StringContent("[" + JsonConvert.SerializeObject(rule, settings) + "]", Encoding.UTF8, "application/json");

    //Have to set this header manually for some reason
    sc.Headers.ContentLength = jsonString.Length;

    //Send the request
    HttpResponseMessage response = await client.PutAsync(ConfigurationManager.AppSettings["CloudflareAPIUrl"]
        + "zones/" + ConfigurationManager.AppSettings["ZoneId"] + "/firewall/rules", sc);

    // return URI of the created resource
    return response.Headers.Location;

}
```

This was pulled from our Git Lab Repo!