**EMCS2000 Introduction to Computer Security**: Module 6 Report

Brian Russel Davis, [brian\_davis@brown.edu](mailto:brian_davis@brown.edu)

### Introduction

Web Application security starts with securing the platform that applications run on. The code used to build the site, the assets that are used by the code and the server that coordinates the communication to the end user constitute an ecosystem, the whole of which needs to be secured. Understanding which platform performs the best under different types of attacks is the key to crafting a well-thought-out strategy for protecting a website and dealing with an intrusion. For Laverty, we have chosen to implement the application using Amazon Web Services. AWS offers a cloud environment where we can configure many layers of security and log all of the activity generated by the application and the users. Access the servers, actions that are taken by the users and actions that are taken by other principals including other applications and scripts will be carefully constructed with the principles of least privilege, logged and monitored.

Securing the front door of the website and preventing as many types of code injection attacks as possible is very important. We also want to make sure that the backend “knows” what is happening in the “front of the house” and is prepared and capable to respond when the organization is attacked. This type of coordination is achieved by creating policies, monitoring activity and thoroughly sanitizing User Generated Content ( UGC ).

### On-Premise Servers vs. “The Cloud”

*“While the term "cloud computing" was popularized with Amazon.com releasing its Elastic Compute Cloud product in 2006,[[1]](#footnote-0) references to the phrase "cloud computing" appeared as early as 1996, with the first known mention in a Compaq internal document.”* [[2]](#footnote-1) So, this technology has been commercially available for over 12 years, in development for over 20. Not to mention this approach to building web applications has seen adoption from organizations ranging from military, banking, healthcare and other sectors that could be said to deal with sensitive data and regulation, there are many organizations who still believe on-premise servers are “safer.” Without taking a side in the debate about safety and management, I would like to propose that it doesn’t really matter where the code for the application runs, as long as there are systems surrounding that application that can control, monitor and manage access to the on-premise server. A bank may have rules that require application code runs hardware “owned by the bank”, but that does mean that all the software used by the bank needs to be built by the bank. Does the bank make its own browser? Is the bank going to build its word processing software or email clients? Is the bank going to design and code its own database? The truth is every organization depends on software they did not build. Running the application code on hardware inside of the bank’s four walls does not make it more secure. However, we can work ***with*** this requirement and set up a system that keeps the application code on the organization's hardware ***and*** uses cloud services to regulation the traffic to and from this server. As a matter of fact, this approach is so popular, there are many ways to implement hybrid systems. Here are illustrations of some popular versions:

<https://www.trendmicro.com/azure/wp-content/uploads/2016/10/Hybrid-Cloud-Deep-Security.pdf>

### Threat Overview

Cross-Site Scripting ( XSS ) is a badly named attack. It implies that there's a script coming from another site that is attacking your site. However, most of the time it has nothing to do with this type of behavior at all. Usually, the attacks are centered around a part of a system, application, plugin or API that does not properly handle input from a POST request. The most simple version of this type of attack occurs when a page that allows user input does not sanitize or escape the input, allowing space for an attacker to submit input that runs a script or forces the application to run a command. So far in 2018 **Common Vulnerabilities and Exposures** (**CVE**®) has listed almost 1700 know vulnerabilities associated with XSS and other types of injection attacks.[[3]](#footnote-2) While only a handful earned a high threat score, meaning they allow an attacker to completely take over a system, the majority of the attacks are still very serious. Most of the vulnerabilities are exploited to expose sensitive user data that might be used in another kill chain aimed at gaining control of a system.

Laverty has a fiduciary responsibility to keep customer information safe, and this duty also extends to internal customers. Protecting against threats like XSS is just the beginning.

### Separation of Duties

Now that we have a good idea of the types of threats we need to protect against, we need to understand the roles people in the organization play in protecting the organization from these threats. Separation of Duties or SoD is a very important concept, especially in the financial world, when developing and deploying applications and handling sensitive user data. By limiting the amount of contact that operations and development staff have sensitive data, we can mitigate against the most common attack vectors, insider threats. We don’t have time in the plan to outline all of the necessary SoD’s, however here are some important points. **Engineering staff should be separated into discrete teams**: :

1. **System Engineers** who have access to internal operations systems ( but not application code, production servers or user data ),
2. **Development Operations Engineers** ( DevOps ) who have access to production servers and user data ( but not application code or internal operation systems), and
3. **Software Engineering** staff that has access to application code ( but not production servers, user data or operating systems)

While these points are often not considered when constructing a site, it should be, because the people build the site and the way they conduct themselves while handling sensitive information, is just as important ( if not more ) than the application itself. In many organizations, the SoD is reflected in the permissions structure. When SoD starts with the roles and assignments in the real world that are defined by HR and the security team, the permissions defined in the access controls make more sense. Furthermore, when the engineering staff is responsible for creating, checking, and deploying their own work, finding mistakes that impact security is harder if not impossible.

### Security Architecture

As mentioned before there are many different parts that need to be secured in an ecosystem that compromises a website and for a website handles sensitive information, these layers are even more important. Building in the cloud comes with the benefits of not having to worry about purchasing hardware and managing the OS on servers, but it has it own worries and complexities that are unique and challenging. From configuring access with IAM roles to crafting smart security policies attached to these roles to granting administrative powers to principals in a Virtual Private Cloud -- There are lots of moving parts in the cloud that need to be carefully thought through. Here are a few key points that must be addressed:

#### Defining Security Architecture in the Cloud

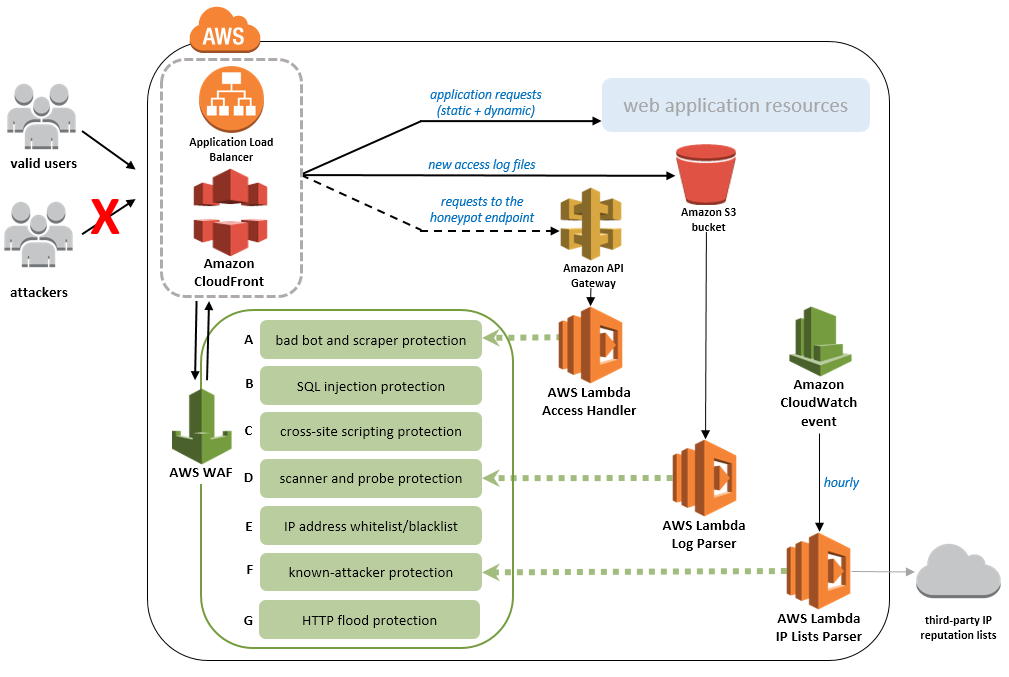
In the cloud, architecture developers are encouraged carefully define relationships between all the parts of the system. Whereas classic monolithic applications consisted of a web server, some application code, and a database, all in the same place, modern applications look very different. In fact, some of the most modern ecosystems do not involve servers ( from the developer perspective ) at all. These modern systems require developers to specify who can access the resources, methods, functions and the limits on the access.

AWS provides developers with a toolkit, namely the AWS Web Application Firewall or AWS WAF:

*“AWS WAF is a web application firewall that helps protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources. AWS WAF gives you control over which traffic to allow or block to your web applications by defining customizable web security rules. You can use AWS WAF to create custom rules that block common attack patterns, such as SQL injection or cross-site scripting, and rules that are designed for your specific application. New rules can be deployed within minutes, letting you respond quickly to changing traffic patterns. Also, AWS WAF includes a full-featured API that you can use to automate the creation, deployment, and maintenance of web security rules. “ [[4]](#footnote-3)*

Building something like AWS WAF from scratch could take months, not to mention managing all of the custom code, agents and policies across the enterprise. Implementing WAF is not simplistic ( and it shouldn’t be ). While the controls are powerful in their preconfigured state, understanding what they are in exactly, helps us determine the implementation details needed to protect Laverty.

Below is an illustration of the system with following components: Honeypot (A), SQL injection (B) and cross-site scripting (C) protection, Log parsing (D), Manual IP lists (E), IP-list parsing (F), HTTP flood protection (G)



The AWS WAF is different than a traditional firewall because it is able to scrutinize activity happening in and around an application at a much more granular level as opposed to a traditional firewall. We could write a book on the strategies referenced in the illustration above, but the defense against these types of attacks boils down to a few important ideas:

1. **Sanitizing input from form fields and any other UGC that comes into the ecosystem from the website.**  These are standards that are established at the engineering level and enforced with vigorous code review, automated testing, and scanning during continuous integration. Errors and omissions related to lack or sanitization are usually the results of honest mistakes by new developers, or forgetfulness by seasoned developers. Congruently libraries or frameworks with premade input fields, upload features or other types of UGC need to be tested with the same level of care.
2. **Monitoring the information in the headers of requests coming into the server along with the user agent type and IP addresses for these requests.** Creating a way to monitor the information in a header and setting up rules to deal with strange or malicious fingerprints will stop most unsophisticated or even moderate level attacks. There are too many types of attacks to mention here, but AWS ( and many others ) have created policy templates to help SysAdmins quickly implement protections from the most common attacks: <https://s3.amazonaws.com/cloudformation-examples/community/common-attacks.json>
3. **Defining acceptable patterns of use for services and resources, with automated responses or actions defined for activity that falls outside the acceptable patterns.** Examples of this may include a limit on login attempts before the service locks up, or simply limiting acceptable file types in an upload interface. AWS CloudFormation, CloudFront and CloudWatch are tools used to design, distribute and monitor cloud ecosystems.

*“CloudFront integrates seamlessly with AWS Shield for Layer 3/4 DDoS mitigation and AWS WAF for Layer 7 protection. In addition, CloudFront negotiates TLS connections with the highest security ciphers and authenticates viewers with signed URLs. You can also use our advanced feature Field-Level Encryption to protect most sensitive data throughout your enterprise, so the information can only be viewed by certain components and services in your application stack. CloudFront also integrates with AWS Identity and Access Management (IAM) to control access, with AWS CloudTrail to log access to your configuration, and with Amazon Certificate Manager (ACM) for automated certificate renewals.” [[5]](#footnote-4)*

1. **Blocking malicious IP addresses and carefully managing a record of the devices a verified customer logs in from.** Users usually log in from the same machines and the same places over and over. Making a user who is signing in from an unknown location complete additional challenges might seem like a hassle but it could stop a serious attack. Detecting the machine’s IP shouldn’t be the only way that the application senses a user is logging in from a new location, however. Behavioral Biometrics is an evolving field, that has some practical use cases for
2. Lastly, **limiting the scope of actions a user, device or any other principal can take in the application is the key to safety.** If attackers are able to bypass the protections on the front end, the application should be able to make a determination about the permissions of the agent making an unexpected change. For example, an API that was designed to authenticate a user should not have permission to create one. Conversely, methods designed to create a user should not have a direct web interface.

### Conclusion

Building a secure application is like building a small city. There are several moving parts and the people directing projects need to communicate. By adopting practices and policies that treat unknown UGC as an assumed threat, and actively monitoring the actions of every principal agent in the system, we can create a ecosystem with built-in safety.

1. "Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta". Amazon.com. 24 August 2006. Retrieved 31 May 2014. [↑](#footnote-ref-0)
2. Antonio Regalado (31 October 2011). "Who Coined 'Cloud Computing'?". Technology Review. MIT. Retrieved 31 July 2013. [↑](#footnote-ref-1)
3. “Security Vulnerabilities Published In 2018(Cross Site Scripting (XSS)).” Apache Http Server Version 2.4.7 : Security Vulnerabilities, www.cvedetails.com/vulnerability-list/year-2018/opxss-1/xss.html. [↑](#footnote-ref-2)
4. “AWS WAF - Web Application Firewall - Amazon Web Services (AWS).” *Amazon*, Amazon, aws.amazon.com/waf/. [↑](#footnote-ref-3)
5. "Amazon CloudFront Product Details". Amazon.com. Retrieved January 4, 2018. To deliver content to end users with lower latency, Amazon CloudFront uses a global network of 107 Points of Presence (96 Edge Locations and 11 Regional Edge Caches) in 55 cities across 24 countries. [↑](#footnote-ref-4)