Software Requirements Specification

for

E-Commerce Honeypot Project

Version 1.0

Prepared by Daniel Mcnair, Eric Audit, Martin Williams

Created 9/2/21

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

Build a variable "honey pot", implement all of the OWASP Top 10 security issues and analyze what happens. We will fix each of the issue. For this project, we will build a believable public-facing simulated ecommerce store and analyze what type of attacks are made on the store.

1.2 Document Conventions

Standard MLS documentation

1.3 Intended Audience and Reading Suggestions

This document is intended to be read by Richard Rauscher, instructor for CIS4935, as such the document will be technically written and expect technical competence to understand.

1.4 Product Scope

Ecommerce site will feature the OWASP top ten vulnerabilities:

A1:2017-Injection; A2:2017-Broken Authentication; A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE); A5:2017-Broken Access Control; A6:2017-Security Misconfiguration; A7:2017-Cross-Site Scripting XSS: XSS; A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities; A10:2017-Insufficient Logging & Monitoring

Web site will be hosted via publicly accessible AWS ec2 instance. Site to include SEIM for monitoring/logging, SQL database for orders and credential storage, and shopping cart functionality. Depending on how our conversation with professor Rauscher goes we might scale this up or down depending on what makes sense for the needs of completing CIS4925 satisfactorily.

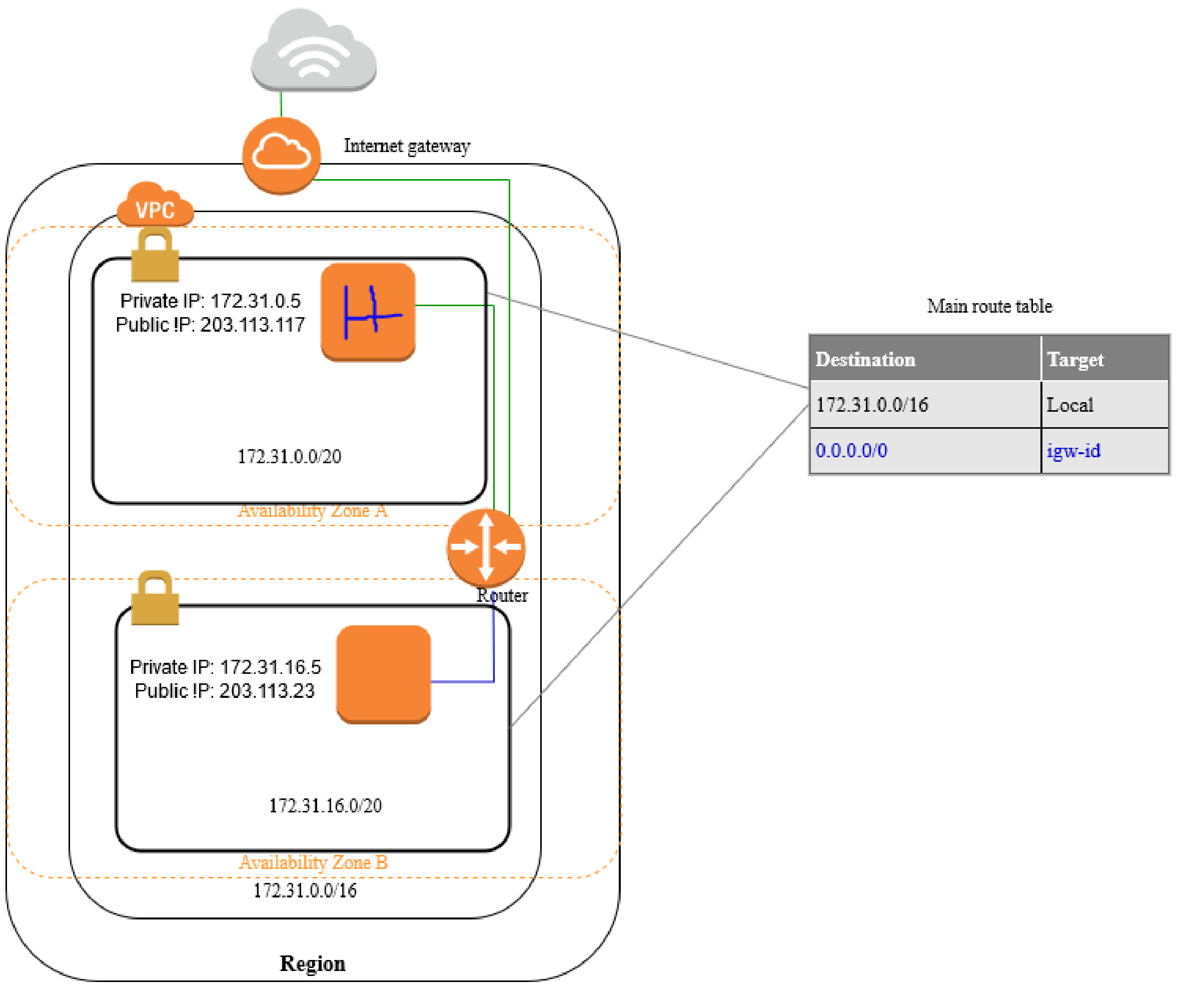
1.5 References

Amazon Elastic Compute Cloud Documentation

2. Overall Description

2.1 Product Perspective

Our honeypot, represented as H in the diagram, will exist inside an AWS Region, likely US east based. We'll configure the Instance to allow all port 80 traffic so bots can hit the site, and we'll lock down the other ports with the AWS firewall. We can use cloudtrail to record 90 day of logging for free, further, we can use Splunk's AWS addon and a limited enterprise account to monitor changes in the instance.



Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	igw-id

2.2 Product Functions

The main function of this web application honeypot is to attract potential threat actors to attack a vulnerable web application in order to analyze threat actor behavior. A secondary function is to take the information gathered by the real world threat actors and use the information to patch the vulnerabilities exploited. This functions as a simulation of a real world environment and shows the complex nature of potential threats, incidents, and defensive techniques.

2.3 User Classes and Characteristics

This product should only be used by us (the students), professor Rauscher, and threat actors (or at least their bots). Hopefully the website is unappealing enough that no one would actually use it. Even though it's supposed to imitate an ecommerce site, hopefully we can avoid having any mechanism that someone could actually pass credit card information or other PPI through.

2.4 Operating Environment

We can leverage AWS to cheaply host our honeypot. We can either spin up a free ec2 instance, deploy ubuntu, and host from there, or depending on final requirements, host it through S3. Likely S3 hosting will not be possible, as this won't allow for server side scripting.

2.5 Design and Implementation Constraints

Our biggest design limitations will be our limited student skill sets and the fixed deadline for completing this senior project. Hopefully we can work out a set of requirements that are realistic for our skill sets, but also meet the needs of professor Rauscher and what is considered satisfactory for the project. Naturally, being students, we have budgetary requirements. We don't think the cost of creating a honeypot will be significant, and we can mostly mitigate this through AWS student credits and free software.

2.6 User Documentation

https://docs.aws.amazon.com/ec2/index.html
as we add linux services and code to the project we can update this with links to the MAN pages for those linux applications.

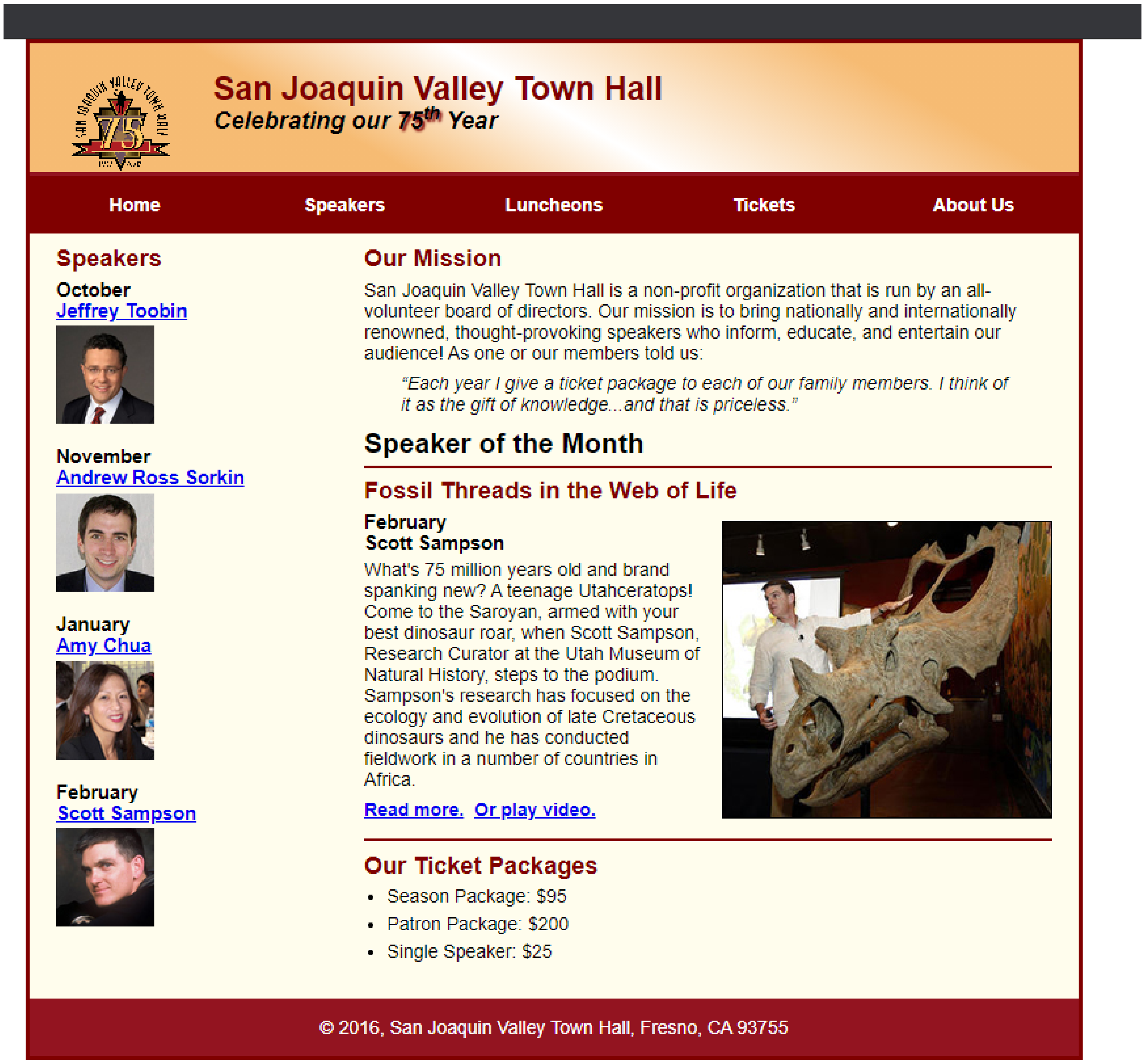
2.7 Assumptions and Dependencies

We assume AWS will be functional for the period of our senior project, and that our linux VMs will have access to our network appliances and the greater web. We also assume that threat actors will attack our low security exposed web page (or at least their bots). The other free softwares we use as the project develops, hopefully they remain free and accessible. Overall there are relatively safe assumptions.

3. External Interface Requirements

3.1 User Interfaces

Here is an old website Eric built as part of his HTML coursework, The e-commerce site will look similar in complexity, element design, and presentation.



San Joaquin Valley Town Hall
Celebrating our 75th Year

Home Speakers Luncheons Tickets About Us

Speakers

October
Jeffrey Toobin

November
Andrew Ross Sorkin

January
Amy Chua

February
Scott Sampson

Our Mission

San Joaquin Valley Town Hall is a non-profit organization that is run by an all-volunteer board of directors. Our mission is to bring nationally and internationally renowned, thought-provoking speakers who inform, educate, and entertain our audience! As one or our members told us:

> *"Each year I give a ticket package to each of our family members. I think of it as the gift of knowledge...and that is priceless."*

Speaker of the Month

Fossil Threads in the Web of Life

February
Scott Sampson

What's 75 million years old and brand spanking new? A teenage Utahceratops! Come to the Saroyan, armed with your best dinosaur roar, when Scott Sampson, Research Curator at the Utah Museum of Natural History, steps to the podium. Sampson's research has focused on the ecology and evolution of late Cretaceous dinosaurs and he has conducted fieldwork in a number of countries in Africa.

Read more. Or play video.

Our Ticket Packages

- Season Package: $95
- Patron Package: $200
- Single Speaker: $25

© 2016, San Joaquin Valley Town Hall, Fresno, CA 93755

3.2 Hardware Interfaces

We'll be using AWS for the hardware.

3.3 Software Interfaces

We'll have the site hosted on an AWS EC2 instance, probably a free Ubuntu one, and the site will have to communicate with a database on the EC2 instance to simulate ecommerce (we might use Amazon RDS if we really want to play with free cloud features). The third component is the monitoring component. Splunk apparently offers a free tier of their SIEM with an AWS addon, using splunk would be ideal, but if there are hidden costs associated we might instead go with ELK stack. Logging and monitoring can be done with AWS's built in Cloudtrail and firewall appliances.

3.4 Communications Interfaces

The honeypot will require access to the internet, and will have to allow the world to connect to it. As such we'll have to allow all port 80 traffic so http communication can exist. The honeypot itself will be a website, so it will have electronic forms for attackers to manipulate and these forms will be submitted to a database to simulate order acceptance. We can either send database traffic to a private subnet in the AWS VPC using AWS's RDS service, or if cost or reliability become a concern as we explore that, we can use a database service on the EC2 instance itself. Both options would be more than sufficient for the honeypot.

4. System Features

4.1 The Website (honeypot itself)

4.1.1 Description and Priority

This is our highest priority, until the website is designed, and the OWASP vulnerabilities are in place, we can't collect any data.

4.1.2 Stimulus/Response Sequences

The site will feature web forms vulnerable to injection attacks, broken authentication, sensitive data exposed, accepts untrusted xml documents, broken access control, security misconfiguration, XSS vulnerabilities, insecure deserialization, known vulnerable components, and insufficient logging.

4.1.3 Functional Requirements

REQ-1: The website should have forms that can be manipulated, and access to a database for manipulation.
REQ-2: Passwords and logins should be passed insecurely so that attackers can intercept and manipulate them.
REQ-3: The structure of the website should allow for cookies to be used for session hijacking.
REQ-4: limited user accounts should be able to see "privileged" parts of the website.
REQ-5: Other requirements TBD after speaking with professor Rouscher.

4.2 The Database

4.2.1 Description and Priority

The database is a high priority, but we can collect meaningful data without it. The database will be essential for collecting attempts at SQL injection and other SQL/Database related vulnerabilities however.

4.2.2 Stimulus/Response Sequences

The website will pass SQL commands to the database without any input sanitization as we collect data on threat actor activity.

4.2.3 Functional Requirements

REQ-1: The database should be vulnerable to injection and manipulation.
REQ-2: Other requirements TBD after speaking with professor Rouscher.

4.3 The SIEM

4.3.1 Description and Priority

The SIEM is a high priority, while not as important as the site itself, it would be hard to collect meaningful data without it. The SIEM will give us insight into threat actor activity and manipulations to the website/database.

4.2.2 Stimulus/Response Sequences

The SIEM will be feed JSON files from Cloudtrail, AWS's native logging service, from there we can build dashboards and run queries to see if threat actors have issued malicious commands.

4.2.3 Functional Requirements

REQ-1: Determine if we will use ELK stack or Splunk (cost factors to determine this, so TBD).
REQ-2: Both options can be deployed in AWS through addons, so we should get visibility into our environment rather quickly.
REQ-2: Other requirements TBD after speaking with professor Rouscher.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

We don't need the honeypot to run well, as such AWS offers a free tier for their EC2 instances, a 2 core processor with 1GB of ram. This is really weak hardware, but if we offload most of the functionality to other AWS services this should be sufficient for running a simple website. If the site is DOS'd or taken offline from heavy bot traffic we might have to play with the configuration, but as long as we collect some useful data and implement fixes then this shouldn't be an issue. AWS' scalability allows changes to the service to be made easily by increasing the hardware allocation for the EC2 instances.

5.2 Safety Requirements

This honeypot doesn't represent a physical danger to anyone, but we'll have to keep our AWS IAM safe from threats, as long as we don't do something silly like store our keys on the ec2 instance this should be fine. Firewall rules will only permit SSH from our IP addresses.

5.3 Regulatory Requirements

The instance will be in the USA, and we will have to comply with AWS's requirements. Amazon doesn't have any explicit rules for honeypots and their red teaming rules give broad freedom.

5.4 Cybersecurity Requirements

The main cybersecurity requirement for the proper implementation of an efficient honeypot is robust logging and monitoring. As a honeypot is not connected to vital infrastructure the security concerns are quite low. Other than robust logging and monitoring for future analysis, for the purposes of this project, protecting the stability of the virtual environments is key. Preventing the EC2 instances from being shutdown or crashing is important to ensuring that vulnerabilities can be patched and logs can be recovered and analyzed.

5.5 Software Quality Attributes

We expect our honeypot to be attacked by all manner of exploits, as is its purpose, specifically we hope to capture data on all OWASP top 10 vulnerabilities, but more refined requirements can be determined after speaking with professor Rauscher. In order to mitigate the OWASP top 10 vulnerabilities a website should:

- A1:2017-Injection: Preventing injection requires keeping data separate from commands and queries.
- A2:2017-Broken Authentication: follow best practices for passwords and credential handling.
- A3:2017-Sensitive Data Exposure: Make sure to encrypt all sensitive data at rest, make sure data in transit is encrypted.
- A4:2017-XML External Entities (XXE): Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data
- A5:2017-Broken Access Control: Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- A6:2017-Security Misconfiguration: A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A7:2017-Cross-Site Scripting (XSS): Preventing XSS requires separation of untrusted data from active browser content.
- A8:2017-Insecure Deserialization: The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types
- A9:2017-Using Components with Known Vulnerabilities: Remove unused dependencies, unnecessary features, components, files, and documentation.
- A10:2017-Insufficient Logging & Monitoring: Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.

OWASP includes other mitigations and recommendations for each of the top 10 vulnerabilities, located here.

5.6 Business Rules

All of the students will have full access to the environment and infrastructure.

6. Other Requirements

Hopefully we can avoid implementing features that allow payments information input on the website. We don't actually trust our ability to run a honeypot and safely handle this information. Be as it may, if we were to implement these features, then we would need to make sure that information is handled within the scope of the law. PCI DSS standards include many recommendations, but until we scope out the project with professor Rauscher I'm only going to cover the highlights.

- Keep cardholder data storage to a minimum by implementing data retention and disposal policies, procedures and processes that include the following:
    - Limiting data storage amount and retention time to that which is required for legal, regulatory, and/or business requirements.
    - Specific retention requirements for cardholder data.
    - Processes for secure deletion of data when no longer needed.
    - A quarterly process for identifying and securely deleting stored cardholder data that exceeds defined retention.
- Do not store sensitive authentication data after authorization (even if encrypted). If sensitive authentication data is received, render all data unrecoverable upon completion of the authorization process.

Full documentation here

Appendix A: Glossary

Will add as we go.

Appendix B: Analysis Models

In previous parts of the document, if too many are included we'll aggregate.

Appendix C: To Be Determined List

Will add as we go.