CS4404

Mission Debriefing Report of Mission 1

Matthew Malone, Nour Elmaliki, Irakli Grigolia

Table of Contents

Introduction

In Shueworld, elections are held every 3 years in order to determine who will be the next Sneaker of the House. Each citizen votes at a local lace.The laces then tabulate their votes and determine who won the election. The candidate with the most laces wins. In the event the laces result in a tie, the two candidates both serve in a united government, known as a “knot.” Citizens cast their vote using computers at their local lace, or with phones which are connected to Shueworld wide web (SWW). Our team was kindly asked to be in charge of Election’s security. We decided to model man-in-the-middle attack scenario, test it on voting infrastructure, and then create a defense that would prevent Mittens Romulan from executing such attacks in the future.

Reconnaissance

Security Goals of an Election System:

1. Confidentiality

Voter information should remain private, as well as who they voted for. If this data was leaked, voters’ identities could be stolen or they could experience retribution for their vote. Data leakage attacks on the voting process(intercepting packets) or attacks against the voter database could result in data leakage. Properly encrypting traffic and protecting the voting database would reduce this risk.

2. Integrity

To ensure confidence in election results, every voter’s identity and vote must be 100% accurate. Stolen or manipulated votes could change the outcome of the election. Session hijacking, sql injection, or other attacks on the voting database could alter the outcome of the election. Protections for session hijacking will be explained later in the paper. Sql injection can be protected through proper sanitization.

3. Availability

Voters must be able to access the election infrastructure during the election. If election infrastructure was taken down, possibly due to a DDOS attack, then voters could be

disenfranchised. This attack might be the most difficult to circumvent, as the voting infrastructure will already be experiencing massive traffic during the election. The government must provide enough resources to serve all voters and fend off malicious traffic.

Section 1: Infrastructure Building

In order to build an effective voting app, we had to consider many things. Node.js was chosen as our main method of developing the web interface due to it being one of the more modern and commonly used tools to do so. We needed a place to store voters' data so we set up MySQL database on Heroku platform. Heroku is a cloud platform as a service supporting several programming languages. After determining the software to use, we began designing the voting infrastructure.

The first question we asked ourselves when working on this mission was: how can we make sure that no voter votes twice? ShoeWorld coincidentally uses a similar Social Security program to the United States, including using the Social Security Number as a unique identifier for each citizen. We decided to use their Social Security Number in addition to their names as the primary way of identifying themselves. This data was then checked with a table in the database of pre-registered voters to prevent just anyone from filling in a name and random number. After the database was configured, we moved onto designing the server.

In the beginning we started with a simple but highly vulnerable HTTP server (Figure 1). The main issue with http is that it is not encrypted so anyone can see information being transmitted with a simple packet sniffer. The list of possible attacks are quite long for example a cache bypassing attack, many TCP related attacks like SYN flooding attack, and any OWASP top 10 attacks just to name a few. In order to make our server more secure we migrated it to HTTPS with HSTS (Figure 1).

For our attack, we decided to focus on a session hijacking attack carried out by a TLS stripping on-path adversary. There are several defenses to session hijacking and TLS stripping that we will explore in the defense section.

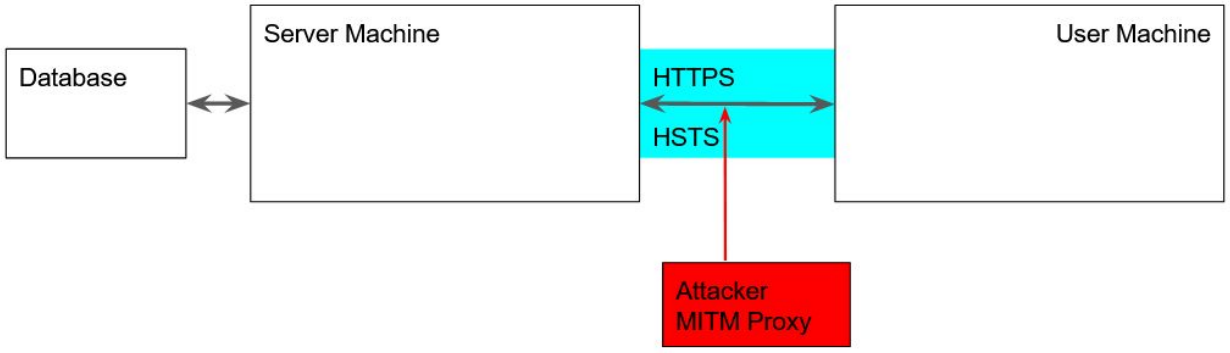


Figure 1. Voting Infrastructure Diagram. HTTPS/HSTS prevent attacks

Setting up the Voting Infrastructure

1. Unzip attached files and read the README
2. Install nodejs (apt install nodejs) OR go to https://nodejs.org/en/ and follow the instructions there to download node js and this will also come with the necessary npm manager.
3. Install npm (apt install npm)
4. Inside the project directory run “npm install”.
5. After dependencies are installed, start the server with “node server.js” (Figure 2).

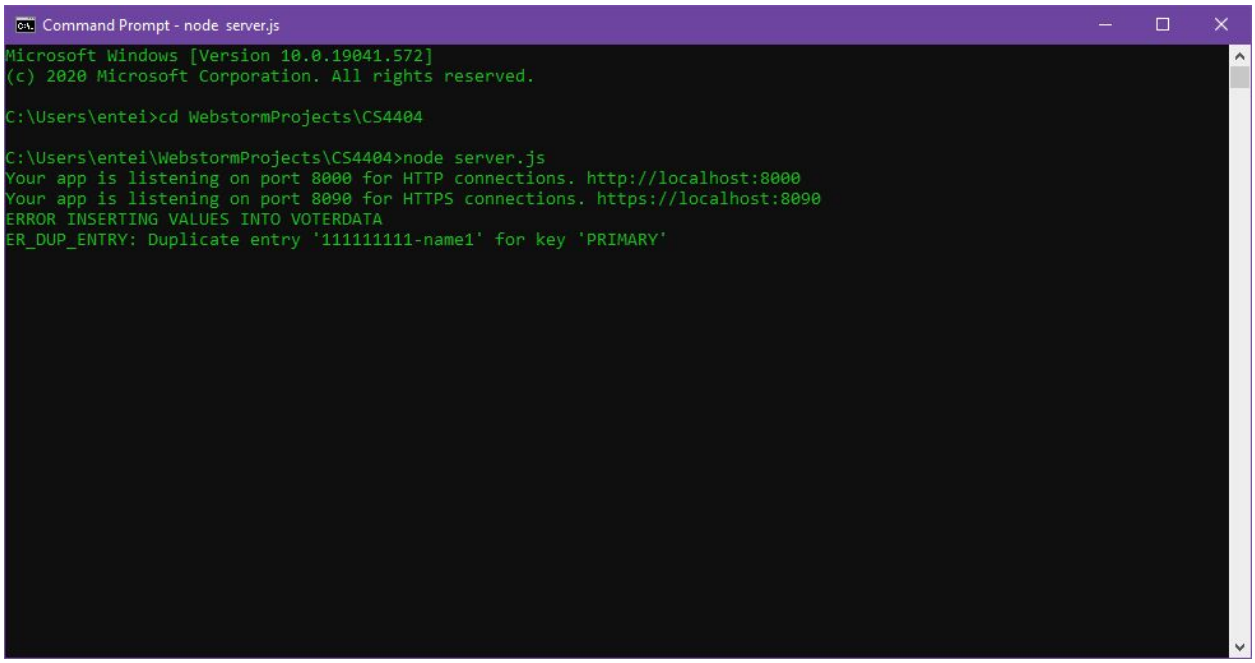


Figure 2. Example of starting the web server. Note the directory is CS4404

Attack Setup

Our attack mimics a situation in which the adversary has gotten on the path of the user and is intercepting their traffic. See figure 3 below for an illustrated example of this. We used two machines, a server and a client. A tool called Mitmproxy was installed on the user machine to mimic the adversary. When the user connects to the server, the “attacker” intercepts the traffic. The attack could be demonstrated by running the proxy and user on separate machines however, getting on-path was not the focus of our attack. This tool can be downloaded from https://mitmproxy.org/ and installed on Windows, Mac OS, and most distributions of Linux with the proper configuring. Documentation for getting this up and running can be found here: https://docs.mitmproxy.org/stable/ however generally this can be summarized into a few steps. Note that in order to keep things simple and fast due to time constraints and for the sake of learning we used this tool on the same machine to capture and modify HTTP and HTTPS traffic. See screenshot 4 and 5 below. Steps for installing MITM proxy to replicate the experiment we did can be found below:

1. Download Mitmproxy and install any dependencies as necessary.
2. Depending on your OS/browser of choice, download and install the certificate by following the instructions here for manually downloading and installing the certificate: https://docs.mitmproxy.org/stable/concepts-certificates/
3. Depending on your OS, go into the proxy settings and configure it to listen to 127.0.0.1 with port 8080
4. Launch the mitmweb application and watch as all your traffic is stolen. You can configure it to search for any traffic on the local machine’s IP to filter the results nicely.

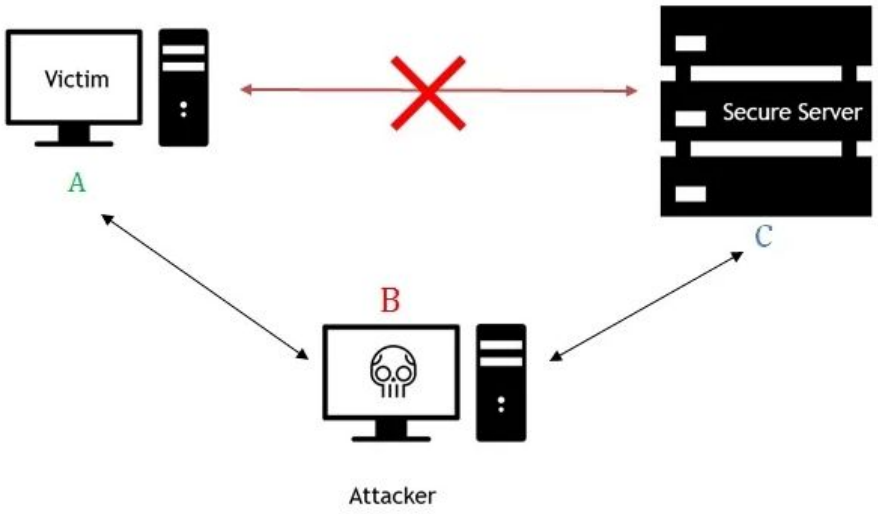


Figure 3. Example of a MITM attack

Section 2: Attacking the Voting Server and Client

We explored the possibilities of several attacks and chose to execute an active on-path adversary MITM attack as our primary focus after deciding it was likely the most damaging to the election by changing votes and stealing voter data. First we covered a basic session hijacking attack using cookie stealing.

Session Hijacking Attack

In most cases when we log into a web application, the server sets a temporary session cookie in your browser to remember that we are currently logged in and authenticated. HTTP is a stateless protocol and session cookies attached to every HTTP header are the most popular way for the server to identify your browser or your current session. It is also possible to modify requests and queries sent from the client to the server if they are only stored in a basic plaintext.

To perform session hijacking, an attacker needs to know the victim's session ID. This can be obtained by stealing the session cookie(aka session side hijacking) or persuading the user to click a malicious link containing a prepared session ID(aka session fixation). In both cases, after the user is authenticated on the server, the attacker can take over the session by using the same session ID for their own browser

session. The server is then fooled into treating the attacker's connection as the original user's valid session.

What Can Attackers Do After Successful Session Hijacking

If successful, the attacker can then perform any actions that the original user is authorized to do during the active session. Depending on the targeted application, this may mean transferring money from the user's bank account, accessing detailed personal information for identity theft, stealing clients' personal data from company systems, encrypting valuable data and demanding ransom to decrypt them or in our case - altering the results of the Shueworld election.

One particular danger for larger organizations is that cookies can also be used to identify authenticated users in single sign-on systems (SSO). This means that a successful session hijack can give the attacker SSO access to multiple web applications, from financial systems and customer records to line-of-business systems potentially containing valuable intellectual property. For individual users, similar risks also exist when using external services to log into applications, but due to additional safeguards when you log in using your Facebook or Google account, hijacking the session cookie generally won't be enough to hijack the session.

MITM Downgrading Attack

When the attacker is in the middle of Victim and Secure Server the attacker can sniff, modify, or drop received packets (see screenshot 7). In addition, it is possible to abuse a feature of browsers to negotiate an older version of SSL/TLS. When a victim wants to send data to a server using SSL/TLS, a client must first go through a handshake to authenticate itself with the server. This handshake starts with the "ClientHello," where the client sends to the server a version of SSL or TLS that it supports, the supported ciphers, and other session data. In older versions of SSL , it was possible to intercept this handshake packet and modify the supported ciphers list to only contain weak ciphers. This is no longer possible since SSLv3 uses a hash in the final part of the handshake, where both the client and server hash and compare sent and received messages. All modern browsers support SSLv3 up to TLSv1.2. Attacker

cannot directly modify any packets sent in the handshake, but he can intercept and drop certain packets. By tricking the browser into thinking that the server does not support a given version of SSL/TLS, an attacker can downgrade the negotiated version.

For example: Client sends "ClientHello" to server. The attacker intercepts and drops the packet. Then he sends “FIN, ACK” over TCP to the server. This terminates the current connection. Client re-attempts connection, sending “ClientHello” with lower version of SSL/TLS.

Protocol downgrade attacks rely on the assumption that an error or termination of the connection means the connection failed due to a SSL/TLS failure. Additionally, in order to be compatible with previous versions of SSL/TLS, a client may attempt multiple connections until a successful connection is made. Therefore, by repeating the protocol downgrade, a middleman can convince the client to negotiate SSLv3 with the server.

SSL Stripping

How is SSL Stripping different from other MITM attacks? The power of SSL Stripping is that the browser won’t display any SSL Certificate errors and the victims have no clue that such an attack is occurring. This attack is also known as HTTP-downgrading attacks, where the connection established by the victim's browser is downgraded from HTTPs to HTTP. For example:

Victim A wants to transfer money from his account using an online banking service and enters the following URL in the address bar of the browser: *www.shuebank.com/online_banking*. In the background, the victim browser connected to the attacker’s machine waits for a response from the server. Attacker B forwards the victim A’s request and waits for the response from the bank server. The connection between B and C is secure, which means that all the traffic that is transferred between them goes through the SSL tunnel. Subsequently the bank’s server responds with the login page that has the following URL:https://www.shuebank.com/online_banking. At this stage, the attacker has access to the login page and can modify the response from the server from https to http. Once this is done the attacker sends the http address to the victim, which results in the browser now being addressed to http://www.shuebank.com/online_banking.

At this point, the victim has access to the internet banking login page with an unsecure connection. From this point onwards, all the victim's requests go out in the plain text format and the attacker can access the data and collect the credentials. While the server thinks it has successfully established the connection.

Section 3: Defense

How To Prevent Session Hijacking

Session side hijacking exists due to limitations of the HTTP protocol. However, session fixation and cross-site scripting (XSS) hijacking rely on the website's implementation of cookies.

To prevent session side hijacking: we used HTTPS to ensure SSL/TLS encryption of all session traffic. This will prevent the attacker from intercepting the plaintext session ID, even if they are monitoring the victim's traffic. Additionally, we used HSTS (HTTP Strict Transport Security) to guarantee that all connections are encrypted. Our attack demonstration will mimic a target that has setup HTTPS but not HSTS.

Session fixation attacks can be thwarted by regenerating the session key after initial authentication. This causes the session key to change immediately after authentication, which nullifies session fixation attacks – even if the attacker knows the initial session ID, it becomes useless before it can be used.

XSS hijacking can be stopped by setting the HttpOnly attribute for the Set-Cookie HTTP header to prevent access to cookies from client-side scripts. This prevents XSS from reading or manipulating the user's cookies.

How to Defend Against a MITM Attack

We decided to make use of HTTP Strict Transport Security (HSTS) in order to strengthen the security of our server. HSTS if set in the response header, tells the browser that it should only communicate using HTTPS instead of HTTP while communicating with the specified domain.

The most important security vulnerability that HSTS can fix is TLS-stripping man-in-the-middle attacks.As mentioned before the TLS stripping attack works by converting a secure HTTPS connection into a plain HTTP connection. The user can see that the connection is insecure, but there is no way of knowing whether the connection should be secure. Back in days when TLS/SSL was not common, there was no way of knowing whether the use of plain HTTP was due to an attack, or simply because the website hadn't implemented TLS/SSL. Additionally, no warnings are presented to the user during the downgrade process, making the attack fairly dangerous.

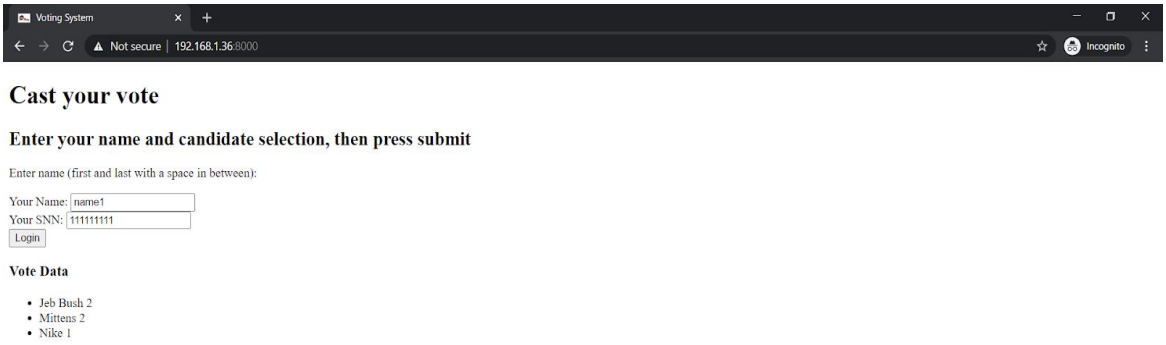
HSTS addresses this problem by informing the browser that connections to the site should always use TLS/SSL. But it is important to note that HSTS header can be stripped by the attacker if this is the user's first visit. In addition, if the attacker is intercepting https traffic, they can still modify results before they get sent to the server for storage/verification. We decided to use HSTS to ensure there was no modification to voter data (see screenshot 6). To protect against this we encrypted the user's cookies with AES256. This is the largest bit size available to the AES standard and is infeasible to attacks using brute force methods.

Conclusion

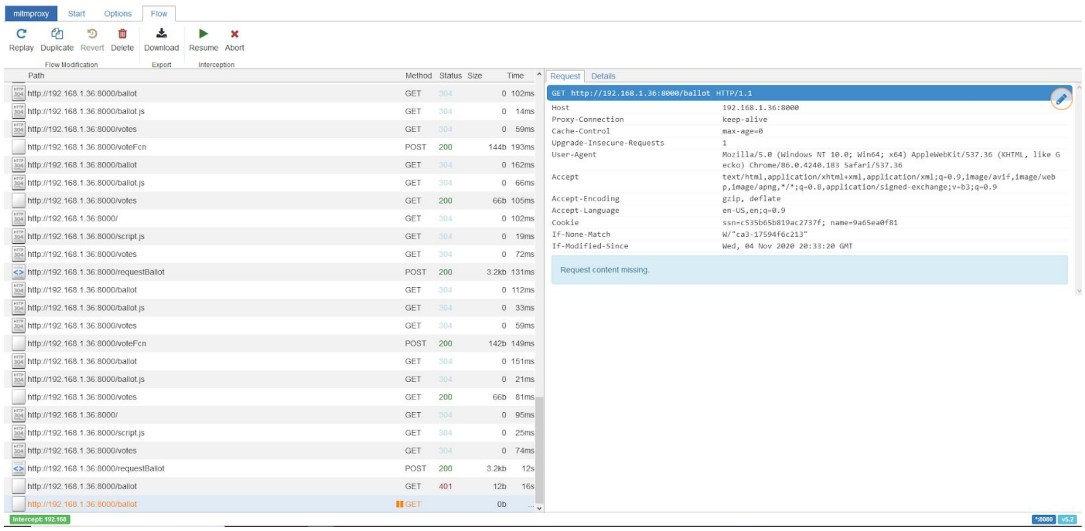
Knowing how important Shueworld's election is, our team did our best to prevent Mitten from using his dirty tricks. We demonstrated a scenario where an attacker could hijack voters' sessions and then implemented a defense against it. In order to prevent MITM attacks we migrated our server to https and used HSTS to prevent SSL stripping. Overall,we think that Shueworld’s election is now secure. Citizens can relax and choose their new ruler wisely so he can make Shueworld great again.

Appendix

Screenshot 1. User logging in

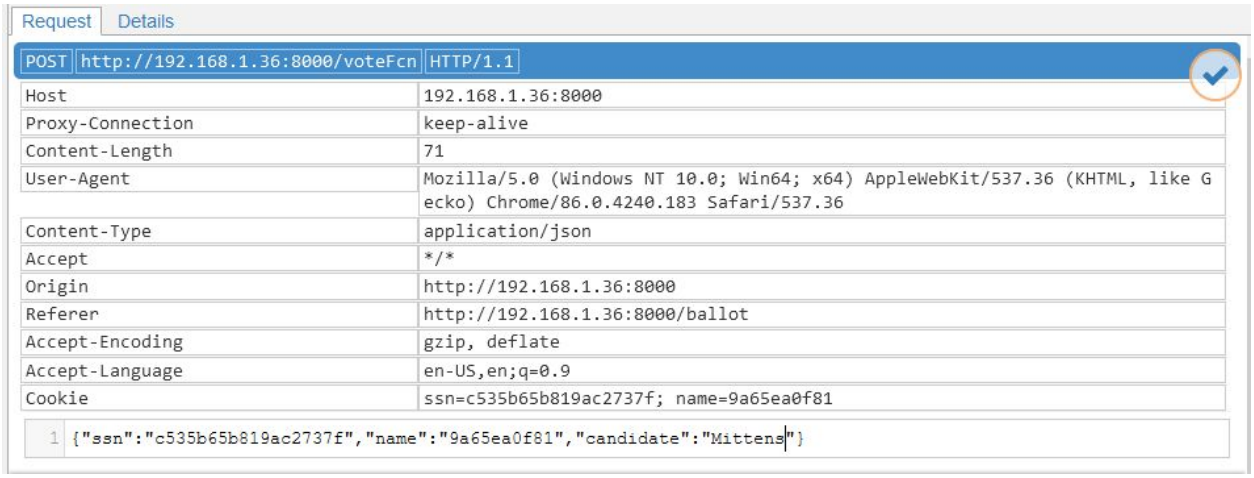


Screenshot 2. Mittens observing traffic



Screenshot 3. Shoeworld Citizen attempting to cast their vote

Screenshot 4. Mittens voting for himself as someone else



Screenshot 5. Voter Impersonated

Screenshot 6. Attempted Voter Impersonated w/ HSTS Enabled