

Mission 1

Jared Grimm, Elsa Luthi, and Roger Wirkala

1. Table of Contents

1. Table of Contents	2
2. Intro	3
3. Reconnaissance	3
3.1 Security	3
3.2 Possible Attack Vectors	4
3.3 Possible Solutions	5
4. Infrastructure Building	7
4. Attack	9
5. Conclusion	15
Sources	16

2. Intro

In Shueworld, they hold an election for their Sneaker of the House every 3 years. Each citizen casts their votes at a lace where the votes are tabulated to determine who won that lace with the most votes in their favor. That candidate who wins the most laces is the winner. If the candidates have an equal number of votes, also referred to as a knot, they serve in a united government. Citizens cast their vote using computers at the lace, or on phones connected through the Shueworld wide web (SWW). Our team was asked to look into the security of Shueworld's voting procedure. In this paper we will talk about different problems that could come about if the voting process is not secure, the possible attack vectors that may be used to penetrate the system, and possible prevention methods. We chose to model a man-in-the-middle attack, test it, and create a defense that would prevent it from happening in the future.

3. Reconnaissance

In this section we outline a comprehensive set of security goals, and the risks of not meeting these goals. We also list possible attacks that could be used on the voting system, as well as solutions to combat these attacks.

3.1 Security

Shueworld is holding their election, and could come face to face with the consequences of inadequate security measures. If an attacker obtains information on citizens who are voting, or alter the outcome of the election, the results could be catastrophic. The Shoes could ultimately lose faith in the government, as well as the security of their information. In a democratic government, like Shueworld's, people must have trust in the government and feel like their voices are being heard. Trust is not an easy thing to earn back, and losing trust in a government may have long term effects on politics. Defective security measures produce more harm than one could think.

To create a secure environment, a system need confidentiality, integrity, authenticity, and availability. Although it may be hard to have all of these features, the more that are included in a system, the safer that system in regards to outside attack. All of these components have consequences if their security goals are not met.

Not having a secure environment puts citizens of a government at risk. A system needs to be confidential in order for information about their own business, or information about their customers to be secure. In the case of Shueworld, the voters should be the only ones who know the information they provide when registering to vote, and who they vote for. If Shoes were to find out that their personal information is leaked there could be a lot of mistrust when providing information to the government. This could pose problems in the future if Shoes decide that voting is not worth potentially having their identity stolen. Shoes also have the right to be the only ones to know who they voted for. If attackers could see which candidate a specific Shoe was voting for they could be threatened to vote another way in the future. When a government's information is not confidential, it puts their citizens at risk and popularizes distrust

in the government.

When a system is not secure it creates distrust from the users. Integrity is equally as important in a secure systems in order to ensure that the information gathered is verified to be true. If Shueworld's election did not have integrity votes could be changed, or there could be multiple votes per Shoe. In situations like these the Shoes would not have faith in their government, because they would not feel like their voices are being heard. The citizens need to be sure that the outcome of the election was based on democracy. When voters feel like there was an equal chance for their candidate to win the election, they are more likely to buy into the outcome of the election. Similarly to the outcome of lack of integrity within a government, lack of authenticity leads to everyone's votes not being equally as impactful. Authenticity can verify that the votes are cast by a real citizen of Shueworld, and they had the freewill to vote on their own. This means that no one is voting for another person, and there is no way to create a fake citizen and vote under that name. There needs to be equality in how much weight everyone's vote receives or it is not a truly democratic government. In order to create the sense that every vote counts, there needs to be authenticity of where the vote is coming from. Availability ties into all eligible voters having their voices heard. Availability means that all Shoes that are allowed the right to vote have the opportunity to vote. In Shueworld, availability could be no denial of service at the actual voting machines. If people were unable to cast their vote they would be upset with the government because their voice would not be heard. Overall, when voters feel like their voices are not being heard they lose trust in the system and are unable to buy into the result.

The main theme of a system lacking confidentiality, integrity, authenticity, or availability is distrust in the government. How can a community trust the Shoes in power if the voters believe that they got into their position in malicious ways. In order to have a productive government, the Shoes need to trust their leaders. Trust is a trait that can take years to earn back and could have many unforeseen consequences when it is between citizens and their government.

3.2 Possible Attack Vectors

There are a couple of attack vectors that could undermine the confidentiality of Shueworld's elections. Two of these vectors are centered around the voting booths and computers at the laces. In any successful and secure election, the voter registration system needs to ensure privacy and voter confidentiality. When a voter goes to the polls they need to authenticate themselves in order to cast a vote. In an online system we assumed they can login with their serial number, equivalent to SSN, an unique identifier, and a form of biometric data such as a shoe print, equivalent to a fingerprint, another unique identifier.

A buffer overflow attack occurs from bad programming practices and insufficient input validation that results in code that is not memory safe. A successful buffer overflow attack allows an adversary to write memory outside of the buffer it was intended to be stored in, leading to arbitrary code execution, or privilege escalation. An attacker could create an image of a fake shoe print and inject that into the machine asking for authentication. This image could contain more pixels than the program is expecting, causing a buffer overflow. With a skilled attack, this vector could be leveraged to spoof authentication credentials, meaning that this attacker could impersonate voters leading to widespread voter registration fraud. This attack could be launched from the voting booth at a computer managed by the lace or from a

mobile device where the attacker has a software tool that allows them to inject the fake shoe print image.

SQL injection is still one of the most popular web hacking techniques employed today. It takes the form of malicious SQL code inserted into web pages that accept user input such as a login page. This SQL code is then run target database and returns the data to the attacker if successful. A successful SQL injection attack in Shueworld could result in the disclosure of voting records. Making a large quantity of voting records public certainly undermines the public's faith and trust in a voting system.

To cast doubt over the legitimacy of an election the integrity can be attacked by altering voter records, changing polling reports, or spreading misinformation. Since a shoe can only vote once per political race, an attacker could intercept a ballot and alter it before it reaches the tabulator server. This type is referred to as a Man in the Middle attack. An attacker inserts themselves into communications between two legitimately communicating hosts. The attacker can then intercept and alter packets between the hosts. There are several methods an adversary could take to deploy this attack. A rogue access point is where an attacker creates a malicious network that can look like a legitimate WiFi network for example. Once a target connects, however, all of the target's network traffic is at the mercy of the attacker. ARP spoofing is another method where an attacker falsely responds to ARP requests to impersonate a legitimate machine by spoofing a MAC address. This method requires finesse through packet sniffing and insertion. DNS spoofing could also be leveraged to launch a Man in the Middle attack. This method is similar to ARP spoofing except that an attacker introduces false DNS cache information to spoof hostname to IP address translations instead of MAC address translations.

Furthermore, attacks on the integrity of the election could come in the form of spreading misinformation. Government websites hosting polling results could be hacked using web hacking techniques such as cross site scripting and SQL injection. Social engineering could also be employed alongside such an attack to sway public opinion and sow distrust for polling stations, the elections, and the government itself.

An attack on the availability of the voting system could certainly cause some public trust issues, but it could cause more chaos than undermining the democratic system. A successful Denial of Service (DoS) attack could prevent legitimate users, in this case voters, from submitting their ballots to the tabulator server to be counted. One way an attacker could launch such an attack is with a botnet comprised of mobile devices. The mobile device botnet could flood the tabulator server with illegitimate traffic and invalid return addresses. The server could be bogged down attempting to process all of the illegitimate requests and not be able to count actual votes. Depending on the strength and size of a botnet, an attacker could quite easily disable the online voting systems at entire laces.

3.3 Possible Solutions

Given the various proposed attacks that may undermine the goals of security, several countermeasures have been evaluated to best protect our infrastructure. The first vector we will address is the possibility for fraudulent or double registration of voters. We could utilize two factor authentication utilizing a salted and hashed database to encrypt the entries of a voter and their unique shoe print, alongside their unique "serial number" which corresponds to a social security number. We could also use stack canaries to defend against buffer overflows from possible binary injections of the shoe print, or serial number. Once authenticated, that user will be added to a database of valid voters for that years election. Which local copies will be stored of this database at every lace, and periodically for cross

referenced to each other for accuracy. If they are already in that database, the new entry will not be duplicated and they will be denied login a second time.

Man In The Middle Attacks are our main vector of concern, as we need to insure all of our network traffic is secure, and no outside sources can see, intercept, or misplace our traffic, thus violating integrity. To accomplish this our ballot will be using Transport Layer Security (TLS) tunnel, enforced through HTTP Strict Transport Security (HSTS) to insure that any attempt to use HTTP will be blocked, and only HTTPS, the encrypted version of HTTP, will be allowed, preventing SSL/TLS stripping attacks, such that attackers cannot see sensitive information or inject their own code into the ballot page. Next to insure the transfers of tallied lace votes make it to the global center untampered with in the middle, we will implement Message Authentication Code (MAC) to act as a checksum where the lace results will be hashed in a cryptographic algorithm with a key K, and that result will be a “tag” or the MAC code. The original message can be sent unhashed, since we don't care if attackers see the results which will soon be posted by the news media anyways. The “tag” will be sent alongside the message, and once arrived at the global tally center, the message received will be rehashed using the same algorithm and key K, and its result will be compared to the received tag. If the tags vary, it will indicate the message was tampered with on the wire, and that that data should be discarded, signalling a detected connection to that lace. Also we can insure only traffic types we permit are on the network by locking down ports. Next we will use the Access Control List of both the sending lace and receiving global tally center, to filter the web traffic it can send or receive based on source and destination IP addresses, source port and destination port, and the protocol of the packet. This will greatly help stop any attempts to intercept and resend data, if any protocol, port, or IP is changed it will be dropped, as well as we can insure no data is ever allowed to travel from the global tally center to the laces. By blocking traffic from all IP addresses except for hardcoded addresses of our lace servers, we can insure no other IP addresses are able to send fake results to our global center, as well as insure that only the secure ports we select for source and destination are being used. If we do not map our IP-MAC addresses statically, and ARP is used, the our system will be vulnerable to ARP poisoning, which can be mitigated on the network using detection tools such as XArp. XArp performs ARP packet inspection on a per-network-interface and can detect active attacks as well as corrupted caches in the network. XArp prevents locally static entries from being overwritten, detecting changes in the local mapping, and checking integrity of ARP packets.

To insure integrity of our votes, backups will be kept both on local machines at the lace level, as well as lace results being shared on a peer to peer network and scattered in pieces. The global tally center will also store periodic local backups every time a new lace is recorded. This will allow for the final results that news sites pull from is always verified against the backup copies to insure no data has been added or manipulated en route.

The HTTP ballot poses areas of entry into our system and must be locked down tightly. To start, we will ensure our input, for say a write in candidate, is sanitized to prevent sql injections. Next, we want to deny the ability to simply script a way to always select a certain checkbox to vote for a certain candidate, so the order of the candidates on the ballot will be randomized using a random seed function such as layered linear-feedback shift registers (LFSR).

To insure integrity of our votes, backups will be kept both on local machines at the lace level. The global tally center will also store periodic local backups every time a new lace is recorded. This will allow for the final results that news sites pull from is always verified against the backup copies to insure no data

has been added or manipulated en route.

4. Infrastructure Building

Ultimately, our team decided that a Man in the Middle attack could be realistic, high-impact attack vector with a practical, effective defense. In order to create a rational and appropriate scenario, our team did substantial brainstorming, and research on weaknesses in online voting. We looked at different attacks, and what attacks could do the most damage to the polling system as shown in *Figure 1* below.

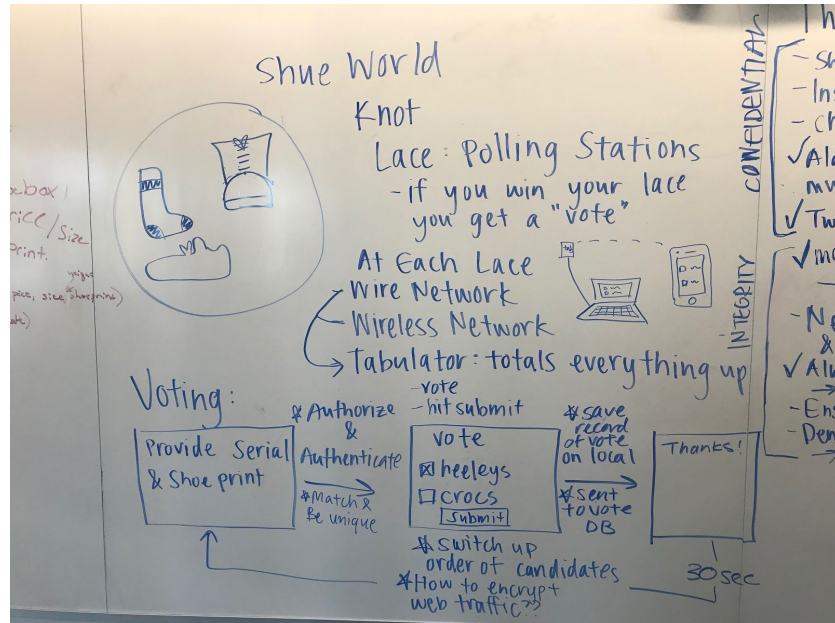


Figure 1: Setup, Elsa Luthi

In order to create an infrastructure that we felt would mimic real life in Shueworld, we created two different frameworks for our project. The first was created in the real world, and was a proof of concept. We created a database, as well as a java program on CCC that pulls that database that updates a webpage. Providing live updates to a webpage models a polling result webpage.

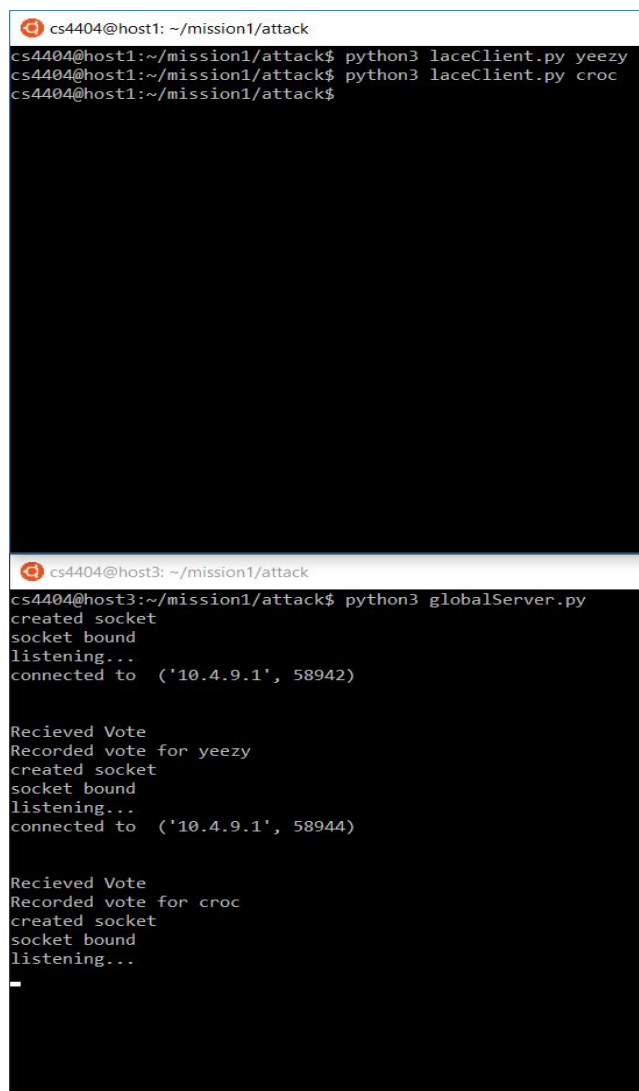
The infrastructure needed to make a web scraper feasible in the VMs was not readily available to use, but there are plenty of web and database solutions ready for implementation and deployment available on the internet.

The first piece that was built was the database to store election results. This was a simple MySQL database deployed through Microsoft Azure. It contained two columns to tally votes for Yeezys and Crocs. We then needed a way to communicate with that database in order to execute queries and gain the current election results. A java program that used the jdbc library was developed to constantly poll and report the results. This program was compiled to a jar file and run on the CCC. Finally, this information needed to be hosted on the internet for the world to see. Roger's personal website hosted through the CCC was used to report the election results live. To retrieve this information from the web page a web scraper was built in python with the BeautifulSoup library. The workflow for this infrastructure is as follows: the jar starts running on the CCC and creates the html for the website at

<http://users.wpi.edu/~rdwirkala/>, then the web scraper python script is run from a machine with an internet connection. As the database is updated, the website and web scraper observe and report those changes.

We chose to implement this infrastructure on the web because it is a powerful example of how the legal portion of our attack can be built and deployed in the real world instead of a protected test network. Since this functionality can only be run on Roger's CCC account, we have provided a video of this working in real time: https://youtu.be/oMF1w_TeZb4.

Our second model was a Python client and a server. These objects can send votes to each other over TCP. This models the laces, and the tabulator. The base infrastructure simply sends votes to the tabulator in plain text. Below, in *Figure 2*, is a screenshot of the infrastructure communicating without any attack or defense mounted.



```
cs4404@host1: ~/mission1/attack
cs4404@host1:~/mission1/attack$ python3 laceClient.py yeezy
cs4404@host1:~/mission1/attack$ python3 laceClient.py croc
cs4404@host1:~/mission1/attack$

cs4404@host3: ~/mission1/attack
cs4404@host3:~/mission1/attack$ python3 globalServer.py
created socket
socket bound
listening...
connected to ('10.4.9.1', 58942)

Recieved Vote
Recorded vote for yeezy
created socket
socket bound
listening...
connected to ('10.4.9.1', 58944)

Recieved Vote
Recorded vote for croc
created socket
socket bound
listening...
-
```

Figure 2: Infrastructure communication

4. Attack

We chose to utilize Man in the Middle attack, where an attacker inserts themselves into communications between two legitimately communicating hosts. We created an adversary posing as a legitimate network in the infrastructure between the client and the server. The adversary is then automated using a scripted web scraper.

The adversary can see all the traffic and manipulate it as it is sent to the global server. It looks at command line arguments that dictate the desired outcome of the election (croc, yeezy, or knot). The adversary looks at the web results from the web scraper, and from this information it makes a decision of what it wants to do with the packet (either change the information or let it go through). If the adversary cannot determine the contents of the packet, it will scramble it regardless of the election status.

The web scraper that the adversary uses in the VMs is a simplified and heavily abstracted version of the web scraper we scripted for the real internet. In the VM, the adversary program reads in data from a txt file stored in the same directory. This approach was the most reasonable given the restrictive nature of the VM environment and the development effort that would have to be undergone to create a web server, client, and database server.

To launch an attack, first connect to the machine at 10.4.9.3 and navigate to the mission1 folder and then the attack folder. Then run the global server python file with python3. Once that is running, connect to the machine at 10.4.9.2 and navigate to the mission1 folder. Once there, run the adversary python file with python3 and make sure to specify the desired outcome as a command line argument (either yeezy, croc, or knot). Finally, connect to the machine at 10.4.9.1 and navigate to the mission1 folder and then the attack folder. Run the lace client python file with python3 and specify who you are voting for as a command line argument (either yeezy or croc). You now have a man in the middle that will selectively meddle with packets that do not meet its goals.

5. Defense

When defending against our implementation of a Man in the Middle attack, we could not stop the adversary from intercepting traffic. Instead, we insured a connected adversary had no way to change the traffic they were sniffing in such a way that they could change the vote to a different candidate, or just cause votes to not be accepted by changing it to nonsense.

In order to prevent an attack, we decided to implement a combination of encryption and authentication. We needed to install gcc (`sudo apt-get install python3-dev`) and python3-dev (`sudo apt-get install gcc`) in order to run a tool library named pycrypto. Next we had to transfer the pycrypto library to the vms over winSCP, and extract the files using: `tar -xvzf pycrypto-2.6.1.tar.gz`. Once unzipped pycrypto had to be built and installed into python, enabling the team to import its tools. To do this, we had to navigate to pycryptos containing folder, enter it and run `sudo python3 setup.py build`, followed by `sudo python3 setup.py install`. Now we were able to add the necessary tools to our client and server.

Before implementing the tools, we had to decide the best combination of them in order to achieve our goals of protecting the integrity of the vote, such that if the vote is manipulated, it will not be accepted, as well as make the action of seeing the vote to change much harder for an automated attack such as ours. We concluded that both encryption and Method Authentication Codes (MAC) would accomplish this. After further research, we found it best practice in this situation to compute the MAC on the plaintext vote, then concatenate the MAC code to the plaintext vote, and encrypt the combination as a whole. We believed it was important for the MAC address to be encrypted due to the fact that two possible MACs would ever be generated with just two candidates, so an attacker may be able to identify the unique MACs, which may also hint data to the plaintext vote. Our scheme of encryption was also carefully selected. We decided to use symmetric encryption, namely Advanced Encryption Standard (AES), as it is faster, and uses 128 bit keys. AES has several different modes of which it can be implemented.

Before encrypting our data, we looked at two different modes of encryption; Electronic CodeBook (ECB) and Cipher Block Chaining (CBC). ECB is deterministic, implying if the same information is put into the algorithm, it will always return the same output. For our project, we chose not to use ECB, because an attacker can identify two encrypted messages that are identical. If an adversary could view messages that were the same size, they would be able to determine who a vote was for due to the repeated characters. Next ECB also isolates attempts at manipulation to one block. If our attacker manipulates part of a block, there is a chance that the data could be recovered, and we will not know that the attacker is present. The mode of decryption we use is CBC, which applies a chaining mechanism that requires the current ciphertext to base itself off of previous ciphertext blocks. CBC also propagates errors, so if just one bit is manipulated, it will propagate to other block, causing clear errors. The very first block is encrypted against an Initialization Vector, or IV. The IV is the same size as the block size for the rest of the encryption, so in our case we used a 16 byte IV, which is randomized at the start of each encryption such that no two transmissions will be the same. The attacker will now not be able to distinguish the MAC or plaintext, or notice any patterns in the votes. The receiver side needs to receive this same IV, so IV is prepended to the encrypted MAC and plaintext message. Since it is a block cypher, all encryptions must be multiples of 16 bytes, so the candidate plaintext must be padded to 16 bytes. This combination of

16 bytes of IV, followed by 32 bytes of encoded MAC (16 bytes) and padded vote (16 bytes) is sent through the adversary to the now protected server. We assume the server has the private MAC and AES Keys previously securely distributed. Decrypting the message involves extracting the various known byte blocks. First the IV is pulled from the first 16 bytes of the received encrypted string. The IV is then used along side the private AES key to decrypt the message. Once decrypted, the first 16 bytes of the decrypted message is the MAC address that was computed at the sender, and the The second 16 bytes is the padded vote response. The first character of the padded vote is checked, to remove the padding according to if its a 'c', remove the last 12 bytes, if its 'y', remove 11 bytes, else the vote will be 'not valid'. The plaintext received vote is then found and a MAC is computed using the same MAC private key as the sender side. The received, and newly calculated MACs are compared, and if equal, the vote is accepted as validated and the global database would be updated and pushed to the web. If found to be wrong, the vote is not accepted, and that lace server is isolated as compromised. Local Lace database results will be retrieved by authorities to determine the real winner.

Below are functions that show the encryption, decryption and authentication:

Libraries needed from pycrypto included

```
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Hash import HMAC
```

Pycrypto generates MAC addresses using the following command:

```
macKey = b'EAD38649A3D3F68E'
mac = HMAC.new(macKey, laceVote.encode())
```

Concatenate MAC onto front of message, know it will be first 16 bytes:

```
macdMessage = mac.hexdigest() + laceVote + padNum
```

Encrypt MAC and plaintext vote using different key:

```
aesKey = b'D83B7A9E7C6F2A10'
iv = Random.new().read(AES.block_size) the block size is 16 bytes
cipher = AES.new(aesKey, AES.MODE_CBC, iv) // note the mode is set to CBC

encVote = iv + cipher.encrypt(macdMessage)
```

Decrypting; first isolate first 16 bytes for iv, next 16 for the Mac, and last 16 for the message where crypto is the full recieved message:

```
iv = crypto[:16]
toDecode = crypto[16:]
padMsg = decrypted[32:]
```

Take the 16 bytes of plain text and analyze the first character to decide how much of the message to pull from the padding:

```
pTextMsg = padMsg.decode()
```

```
if pTextMsg[0] == "c":  
    vote = pTextMsg[:4]  
    vote = vote.encode()  
elif pTextMsg[0] == "y":  
    vote = pTextMsg[:5]  
    vote = vote.encode()
```

Generates hmac on vote assigned above and compare:

```
calcMac = HMAC.new(macKey, vote)  
if (mac == calcMac.hexdigest().encode()):
```

We did three tests for our defense. In our first scenario there is no attack from the adversary. This test, shown in *Figure 5* below, just illustrates that our lace can take a vote for ‘croc’ calculate a MAC, encrypt it, send the result to the global server which will then decrypt it properly, showing ‘croc’ and it will calculate and compare MACs, which upon success, updates the database. The second test we performed, shown in *Figure 6* below, is an example of how our solution could protect against a Man in the Middle attack. In this attack the lace votes ‘croc’ calculates a MAC, encrypts it, and the adversary, cannot read the encrypted data, and flips it to a random message, which is received at the global center, and determined to be an invalid vote that matches none of the candidates and fails authentication as well. Finally in *Figure 7*, we show that if the attacker was somehow able to crack the encryption of a vote of ‘croc’, and replace the vote with his candidate ‘yeezy’, it would still fail at the global center since the attacker would not have the private MAC key and the recieved, or even an attempt at spoofing a MAC address would fail and cause the global center to not accept the vote.

```
cs4404@host1: ~/mission1/both
cs4404@host1:~/mission1/both$ python3 protectedLaceUpdated.py croc
Calculated MAC
e5481113bfbf4deb692bbbeda7f31cb8
Encrypted Vote and Signature
b'\xbdg\x1f\xe615\x8d\x94#\xe9eH)\xf6\xe5|M\xd8?\xed\xf0\x06\xda\xe1\xd3k\xcd\x97\x7f\x83\xc4\x99\x7f\x88,\xa0\xaf\x0b\x02\xd9Ltif\xb0\xbb\xd1]e+\xd9\xe0\x1a\x83\xcf\xbe\xdb\xe0\xc3\xc7\x10\xee\xf9\x19'
cs4404@host1:~/mission1/both$

cs4404@host3: ~/mission1/both
cs4404@host3:~/mission1/both$ python3 protectedServerUpdated.py
created socket
socket bound
socket is listening
connected to ('10.4.9.1', 58946)
Encrypted Packet Received
b'\xbdg\x1f\xe615\x8d\x94#\xe9eH)\xf6\xe5|M\xd8?\xed\xf0\x06\xda\xe1\xd3k\xcd\x97\x7f\x83\xc4\x99\x7f\x88,\xa0\xaf\x0b\x02\xd9Ltif\xb0\xbb\xd1]e+\xd9\xe0\x1a\x83\xcf\xbe\xdb\xe0\xc3\xc7\x10\xee\xf9\x19'

Encrypted Vote
b'M\xd8?\xed\xf0\x06\xda\xe1\xd3k\xcd\x97\x7f\x83\xc4\x99\x7f\x88,\xa0\xaf\x0b\x02\xd9Ltif\xb0\xbb\xd1]e+\xd9\xe0\x1a\x83\xcf\xbe\xdb\xe0\xc3\xc7\x10\xee\xf9\x19'
decrypted message
b'e5481113bfbf4deb692bbbeda7f31cb8croc716c8ad64039'
Decrypted MAC
b'e5481113bfbf4deb692bbbeda7f31cb8'
Decrypted Vote
b'croc'
Calculating MAC on Vote
e5481113bfbf4deb692bbbeda7f31cb8

Comparing MACS
Vote Successful
```

Figure 5: Defended Infrastructure, without Attack

Sources

Security

<https://pdfs.semanticscholar.org/97e8/1919b1a8cece7d7676b4e142dcf830e947ac.pdf>

Chapter 1.1 to 1.5: Pfleeger and Pfleeger, "Is there a security problem in computing?", Security in Computing, 4th edition.

<https://www.merriam-webster.com/dictionary/democracy>

Attack

<https://www.hackingtutorials.org/exploit-tutorials/buffer-overflow-explained-basics/>

<https://www.us-cert.gov/ncas/tips/ST04-015>

<https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/>

Defense

https://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol

<https://www.cyberciti.biz/faq/iptables-block-port/>

<http://www.pearsonitcertification.com/articles/article.aspx?p=1868080>

<https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/infosec-guide-defending-against-man-in-the-middle-attacks>

<https://docs.oracle.com/javase/7/docs/api/javax/crypto/Mac.html>

<https://www.quora.com/What-is-the-difference-between-SSL-and-SSH-Are-they-both-just-a-way-to-safely-access-a-remote-computer-through-encryption-Do-they-also-transfer-data><https://www.bbc.com/news/10123478>

Infrastructure

<https://nvotes.com/process/>

https://cs.stanford.edu/people/eroberts/cs201/projects/2006-07/electronic-voting/index_files/page0001.html

<https://www.crummy.com/software/BeautifulSoup/>

Attack

<https://ieeexplore.ieee.org/abstract/document/4768661>

http://delivery.acm.org/10.1145/1030000/1023662/p90-meyer.pdf?ip=130.215.172.248&id=1023662&acc=ACTIVE%20SERVICE&key=7777116298C9657D%2E71E5F5E88B9A3E17%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1541521621_9e523a1c27d9a1841aeb7d2d27ce1f0d

Defense

<https://patentimages.storage.googleapis.com/79/01/7f/1e5bad8ae25ae5/US5438622.pdf>

<https://adayinthelifeof.nl/2010/12/08/encryption-operating-modes-ecb-vs-cbc/>

<https://crypto.stackexchange.com/questions/202/should-we-mac-then-encrypt-or-encrypt-then-mac>

Pycrypto tar files <https://pypi.org/project/pycrypto/#files>