Team 2
CSCI 430
Prof. Mirkovic
13 November 2023

CTF #1 Report
Team 2

CTF1 reports are due 11/13. Please have them as shared Google docs with all team members. Each team member should specify what they did before the attack phase (so in the dev phase) and what they did during the attack phase. You can also list what you noticed about attacks on you or about the server you were attacking. Then share the Google doc with me.

Before Attack Phase:

Katie

- Before the attack phase I worked on getting our server up and operational. I started by creating a simple flask server with one app route that could just take in url parameters and return them back. I tested this locally on localhost:5000. Then in order to make the server available publicly I installed apache. First I tested that I could access the apache default page using w3m. Then I configured it to host my server instead of the default page. I edited the apache2/sites-enabled/000-default.conf file and flaskapp.wsgi file to configure and link everything. Once that was working I started implementing the logic necessary to run the fake "bank". First I imported python requests and created app routes that took in the proper parameters for each .php instruction. Next, I imported sqlite3 and created a database file. I started adding simple logic to add users, update balances, etc. I used pycryptodome to sha256 hash passwords before storing them in the database. I also used secure database calls that used (?) instead of concatenation to ensure that our server was not vulnerable to SQL injections. Once this was working I added the logic for login and cookies. I decided to use python sessions instead of just plain cookies since this meant our cookies would be encrypted. I learned how to save and use cookies using curl and tested this. Once this was working our server had everything it needed to start being attacked. While this was the bulk of what I did I also had some miscellaneous contributions. I created a step by step instruction document for my team that detailed how to find the server code, modify it, and test it Server Instructions . I also spent some time trying to figure out how to run the server over https, however we switched back to http since it was having some issues.

Pooja

- Once the server was setup, I experimented with using iptables to filter and monitor the network traffic. Initially, I attempted to use the Pusher library via a Python script but discovered that it required a GUI-based server to run its dashboard on. After that, I research various iptables rules to allow traffic into ports 80 which is the http port and 23 which allows tcp and udp connections. I also attempted to block other incoming traffic except for the expected tcp/udp responses that were incoming from port 53. This was initially a problem with our server because the filters were too extensive and caused significant slowdowns in the server's operations. Once the filters were reset and redeployed, there were less issues with the speed of server response.

Sohee

- Before the attack phase, I researched how to detect and close backdoors to create a script that can detect backdoors. I also researched ways to make lists of detected and closed backdoors and a way to run the backdoor script automatically in the background. For more information: Backdoor Script
- I searched about SQL injection and how to use SQL injection as a way to attack the server. I also found out some SQL injection lines to try duing the attach phase, that was injecting codes to register or login with particular usernames and passwords.

Natasha

- I researched password cracking and SQL injection methods to prepare for the attack stage. I compiled a list of potential SQL injections we could try on Bank 1 to reveal sensitive user information. I also tested these injections on a copy of our own server to ensure the blue team's username/password verification denied potential injections. I downloaded cracking softwares such as John the Ripper and tested them out on example password hashes, though we did not ultimately focus on password cracking for CTF 1.

Georgia

- Before the attack phase I learned how to change our IP address so that we could use multiple IPs for our attack.
- Natasha and I got t-shark set up to use on the red and blue servers, and tried to detect http traffic on the blue server, as those requests should be unencrypted. We werent able to see much however because there were a lot of other requests going on in the network.
- I also learned how to send requests to the other server and compiled a list of commands to try

Jaime

- Due to misunderstanding/miscommunication I was under the impression that I was part of a different team and unfortunately joined the team once most of the development phase was completed. The team I was a part of failed to inform the professor that we were going to work together so I was placed into team 2 without me knowing (I didn't see the email sent by the professor right away). I still made an effort to study the code that was developed and pieced together by our team to ensure I can be supportive during the attack phase. I spent some time looking into flask which was used during the development phase, as well as tshark which was used during the attack phase.

During Attack Phase:

Natasha

- I first attempted to use every SQL injection method that I had recorded, most of which threw errors. As the attack phase progressed, I tried researching new methods to inject into the attacker's system using characters such as %20 in place of spaces, but the bank did not recognize the commands as valid usernames or passwords. I then helped set up tshark on the server so that we could monitor incoming/outgoing traffic to and from the bank. After eventually being able to see what traffic came through, I researched filters and tried out different ones on tshark to see if I could capture valuable packets that could give me users' login information. Though I believe I was able to filter these packets into ones where login information was being sent, I wasn't sure how to decrypt them to get information from them. I also scanned our server and the other server using nmap for backdoors, but I didn't notice any. We later learned from the professor that many were open near the end of the CTF, so I noted that for the next one.

Georgia

- At the begining ot the attack phase I ran nmap to try to find back doors, but I was unsucessful in finding any so we thought that the other team must be closing them all immediately. However, I think this was not the case and that the back doors had not been opened yet.
- We were able to collect packets from t-shark but not able to decipher any information from them
- Our successful exploits came from providing malicious inputs to the server. We quickly found that the server didnt have protections against depositing and withdrawing negative amounts of money. Therefore we could get around restrictions intended to prevent negative account balences by depositing a negative amount of money. Another issue was that the server wouldnt accept a decimal amount of money, and would crash/ fail if you tried. I also found that by depositing a lage enough amount of money (309 9's) the server would crash.

Jaime

- During the attack phase, I primarily focused on supporting the red team in finding exploits in the opponent's blue team network.

  Initially, I devoted my time to making simple requests using the instructions listed in the CFT assignment write-up.

  I found that registering a user worked fine, but any other request required me to be a super user, forcing our red team to run superuser do (sudo) to interact with the web application. I'm unsure if this was by design, but it's worth noting.

One behavior that we found by simply interacting with the application was that if a user had a zero balance, they would be unable to withdraw any money, as expected; however, if a user had a zero balance and withdrew a negative amount, the user would have that amount (but positive) deposited to their account, which we would consider to be unexpected behavior.

Another attack that I attempted but was unsuccessful was SQL injection attack. I tried several different methods, but the only behavior I got out of these attempts was that the terminal would hang with the character '>', forcing me to terminate the process to continue. I suspect the opponent blue team was protected against SQL injection by escaping special characters or using some prepared statements.

Finally, another attack I attempted was a Denial-of-Service (DoS) attack. I created files to run register, login, deposit, and withdraw commands by opening 8 terminals and running the commands using for-loop and the iterating integer as a unique identifier for usernames/passwords. I did not notice any unusual behavior with the exception that registering a new user was a significantly slower operation compared to the others.

Katie

- During the attacking phase I researched some ways to sniff network traffic but was mostly unsuccessful. I also aided my teammates in some of their testing for various red team attacks by creating a duplicate server for testing attacks on. Finally, I helped my teammates access and interpret the apache error logs whenever we had issues with the server.

Pooja

- During the attack phase, I experimented with different lines of attack on the red team server and methods to defend our server. I researched how to use nmap to scan for backdoors but was unsuccessful in implementing the proper way to close the opened ones. In terms of defending, once we discovered that the red team's accounts were able to malfunction when we attempted to deposit a negative balance. I implemented input validation for our server where you could not deposit negative amounts of money. Additionally, when you were withdrawing money, you could not withdraw negative amounts of money or overdraw the account. We also tried to prevent buffer overflow attacks by setting limits on the maximum amount of money able to be deposited and withdrawn in any given transaction. Another defensive tactic was to store hashed passwords that were also salted for additional security to protect even easily guessable passwords. I used a randomly generated number as our salt value and added it to the original password before hashing. This eventually ended up creating issues with the way our database was configured so we didn't end up using it but the concept of securing passwords effectively is a relevant skill that we will revisit in CTF #2.

Sohee

- During the attach phase, I tried the SQL injection lines I prepared before the attach phase. At first, the registration seemed to work, but as I tried to request the money, an error occurred. Later, I researched other SQL injection method that were used to attack servers.

Blue Team: CTF 1 Server Code

Security Features:

- Hashed passwords instead of storing as plain text
- Used Pythons session library with private key instead of plain unencrypted cookies
- Safe SQL injections (?)
- Balance checks to avoid withdrawing negative amounts or more money than you have (may not be in the linked code backup since we added during attacking phase)
- Error handling to handle duplicate usernames

Reflections:

I think our server was okay, but could have had some improvements. I think the three main things would have been better logging, a better system for modification, and https support.

On the logging end, I think having a database that tracked every request the server received along with what user was logged in, what IP they were using, time stamp, whether or not it was a success etc. would have been very helpful. I know we lost some points to the red team, but without that logging, I'm not actually sure what all of those points were from. Better logging would have helped us detect problems more reliably.

Another thing that I think would have been very beneficial would have been a better system for server modification. We had our server ready pretty early but when it actually came to deploying we ran into issues. These ended up being due to small changes we had made after the original round of testing. If we had a better system that allowed us to make changes on a tester server, run that server through premade unit tests, and only if it passes then deploy it I think it would have made things run a lot smoother.

Finally, while we tried to get https working, it was having some problems, and since it was during the scoring phase we didn't want to keep accidentally messing up the server so we ended up not implementing it. Although, I don’t think our opposing red team did try this, in theory they likely could have sniffed the requests being made to get important information like username, password, and balance from the request parameters and responses which we did not want.

While all three of these things would have been nice, at the end of the day it was our team's first time ever making a web server. Even getting some of the simple things up and running required a lot of new learning and debugging for us. Now that we have that background though I think we will be able to dive a lot deeper into more advanced security in the future. I think it was a really great hands-on experience with web development, and it was so cool to see the server come together!

Red Team: Red team Exploits

Exploiting Bank 1: Our red team first tested if there were any illegal actions we could perform on Bank 1, or if their bank gave us any strange messages it shouldn't have.

1. We were initially able to deposit and withdraw negative amounts of money, which allowed users to have negative balances. We could also withdraw an unlimited amount of money ("stealing from the bank").
2. An internal error was thrown when we withdrew or deposited a decimal amount into a user's account.
3. We attempted to deposit a huge amount of money to see if we could cause an error or overflow. We did 'curl -b /tmp/cookies.txt -c /tmp/cookies.txt "http://blueserver/manage.php?action=deposit&amount=999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999"' (309 9's in a row) and got a server error. We weren't sure whether this was a buffer overflow or some other issue.
4. Sometimes, when we logged into an account and attempted to deposit or withdraw funds, the user was not detected as logged in. This was a strange issue because not everyone on our team encountered this error when on our redserver.
5. At one point, we could log out but still be able to deposit into the original account, even when logged into a different account. We are not sure whether it was a cookie-related issue on our end or the bank's end.

Observations: We created potential plans of attack based on these previous observations and some others.

1. We saw that Bank 1 used http instead of https, meaning passwords were being sent in plain text. So, we considered how we could get into legitimate users' accounts.
2. After lots of trial and error, we were able to detect traffic on the server using tshark, though we were overwhelmed by the amount of traffic going to and from it, and had to work to filter it to see what data was important.
3. We scanned for potential open ports using nmap on the blueserver. We didn't find any the first few times we scanned, so we didn't attempt to exploit any. However, we later found out there were open ports on Bank 1 after the CTF in class.

Attack Methods

1. We attempted various SQL injections on the server but they were fully protected against.

    - Couldn’t add spaces to cURL query
    - Then we tried using %20 or %90 instead (spaces or tabs) but so far it defends against invalid credentials
2. We then tried to sniff packets using tshark, but we weren’t able to directly decrypt the information in it

Standard Requests:

```
sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt
"http://blueserver/register.php?user=USERNAME&pass=PASSWORD"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt
"http://blueserver/login.php?user=USERNAME&pass=PASSWORD"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt
"http://blueserver/manage.php?action=deposit&amount=AMOUNT"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt
"http://blueserver/manage.php?action=withdraw&amount=AMOUNT"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt "http://blueserver/manage.php?action=balance"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt "http://blueserver/manage.php?action=close"

sudo curl -b /tmp/cookies.txt -c /tmp/cookies.txt "http://blueserver/logout.php"
```

Overload Server:

```
curl -b /tmp/cookies.txt -c /tmp/cookies.txt
"http://blueserver/manage.php?action=deposit&amount=99999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999999
9999999999999999999999999999999999999999999"
```

Change IP:

```
sudo ifconfig eth0:2 1.2.95.22 up
```

- eth0:X is the way you use that specific ip address
    - EX: curl --interface eth0:2 -b /tmp/cookies.txt -c /tmp/cookies.txt "http://blueserver/manage.php?action=deposit&amount=1000"
- You can make the ip anything that isn't currently used
- X is the ip number basically, so we have 2 and 3 in use

Tshark:

```
sudo tshark -f "tcp" -> specifically get tcp packets
sudo tshark -f "udp port 53" -> dns packets
sudo tshark -f "udp port 53" -T fields -e ip.src -> ip sources of dns
tshark -r your_packets.pcap -Y "dns.qry.type" -T fields -e ip.src -e ip.dst -e dns.qry.type -e dns.qry.name
```