Installation/Configuration

Much of this is repeated information from the bank installation documentation, as it was used to create the first instance of this project.

Installation

Ubuntu 20.04 Regular Installation

SSH

```
Sudo apt-get install openssh-server
```

To create a new RSA key - Sudo ssh-keygen -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_host

This command writes over the generated key created by default. This is to prevent man in the middle attacks from people with the same .iso file/vm available as you.

PHP

```
Sudo apt install php7.4-cli
```

To check: php -v (Should return version of PHP installed)

```
Sudo apt install php libapache2-mod-php php-cgi php-cli
```

The next command is necessary for the server to run php within the files, and therefore, the database

```
Sudo apt-get install php-mysqli
```

Apache

```
Sudo apt install apache2
```

To check: apache2 -v (Should return version of Apache installed)

Necessary Files

The files can be downloaded from ******************.

Go to the MariaDB section and complete all of the steps before continuing in this section
All but the phish_pond.sql file should be saved in /var/www/html. Phish_pond.sql should be saved in the documents folder or a new directory.

You’ll need to create two new directories for the logs and malware files.

```
/usr/local/phish/logs
/usr/local/phish/uploads
```

Start a php server with: sudo php -S localhost:81

Then go to the /var/www/html/ directory to view all of the files that have been moved.
Use a browser to test the placement by searching for localhost. A login page should appear.
Don’t enter anything yet because we are not finished configuring the server.

The next step is to edit the connection.php file in this directory. This file is used in several other files and contains the MariaDB connection information.
The $dbserver is default - 127.0.0.1
The $dbuser is the user you created in the MariaDB section that was given all access to the phish database.
The $dbpassword is the password connected to the user above
The $dbname is pond by default unless changed in the phish_pond.sql file.
This file is then correctly configured to use and connect to the database.

MariaDB

```
Sudo apt install mariadb-server
Sudo mysql -u root (Should log into MariaDB and show version)
```

Database

Phish_pond.sql is used to create the database:

You can change the credentials of the local database user ‘moleary’ in the INSERT INTO USER query at the bottom of the file.

Use the command ‘source’ and then the complete file path to phish_pond.sql. In this scenario, these files were extracted into the Documents folder of ceo’s account.

```
Source /home/ceo/Documents/phish_pond.sql
```

User

```
CREATE USER ‘ceo’@’localhost’ IDENTIFIED BY ‘password1!’;
GRANT ALL ON pond.* TO ceo@localhost IDENTIFIED BY ‘password1!’;
```

OR

Only the second command if you did not already create the user ceo. Granting privileges will create the account if not already created.

Now go back to Phish Server Files and complete the rest of that section

HTTPS Certification

Self-signed certificates

```
Sudo a2enmod ssl
```

```
Sudo a2ensite default-ssl
```

```
Sudo service apache2 restart
```

For the next command please change hostname to your server's name. This command creates the private key, which should not be shared.

```
Sudo openssl genrsa -out /etc/ssl/private/hostname.key 2048
```

The next step is to create a self-signed certificate using the private key created above. The other option is to create a certificate request then have the CA sign it and return it.

```
Sudo openssl req -new -x509 -days 365 -key /etc/ssl/private/hostname.key -out /etc/ssl/certs/hostname.crt
```

This command will then prompt you to enter information pertaining to your server and certificate.

CA signing certificates

For simplicity, the bank server will also be the CA. We will go through creating and signing certificates

To create a signing request:

```
Sudo openssl req -new -key /etc/ssl/private/hostname.key -out /etc/ssl/hostname.csr
```

To sign requests, a directory should be created for CA usage. To do so:

```
Mkdir -p /etc/ssl/CA/certs
Mkdir -p /etc/ssl/CA/crl
Mkdir -p /etc/ssl/newcerts
Mkdir -p /etc/ssl/CA/private
Chmod 700 /etc/ssl/CA/private
```

The above commands create the directories and alter the private key folder to only allow root access. The command below creates a CA private key. This command will also allow you to create a password for the private key. This password will be used every time you wish to sign a certificate with this key.

```
Openssl genrsa -aes128 -out /etc/ssl/CA/private/CA.key 2048
```

Next the CA will create a public certificate to be distributed to all servers and clients to trust other servers whose certificates are signed by this CA. To do so, enter the command below. The challenge password will also appear. If you created one with your key, enter it now.

```
Openssl req -new -x509 -days 365 -key /etc/ssl/CA/private/CA.key -out /etc/ssl/CA/certs/CA.crt
```

Next, you will sign the .csr that was created above with the CA.crt and CA.key to create a signed certificate. To do so:

```
Openssl x509 -req -days 365 -in /etc/ssl/hostname.csr -CA /etc/ssl/CA/certs/CA.crt -CAkey /etc/ssl/CA/private/CA.key -out /etc/ssl/CA/newcerts/testerbank1.crt
```

Afterwards, open the browser you will be using. Click on the setting menu in the upper right hand corner (three horizontal bars). Then choose the Preferences option. Click on Privacy & Security, then scroll all the way down to the bottom of the page until you see the header Certificates. Click on View Certificates. This will create a pop up menu. Click on import then find the ca.crt that was created previously. This allows the browser to trust all sites that are signed by this CA.crt.

You will have to change where Apache looks for the real signed certificate and the private key. To do so, edit the file /etc/apache2/sites-available/default-ssl.conf. The entries SSLCertificateFile and SSLCertificateKeyFile must be matching to where hostname.crt and hostname.key are.

If you were to complete the CA steps from another computer and then import ca.crt and the signed hostname.crt file, you would have a secure, encrypted connection from any system that downloads and imports the ca.crt. Otherwise, a security warning will appear on the screen.

File Descriptions

Authentication.php

This file authenticates or denies users after credentials are entered into login.php. If the users credentials are accepted, either through LDAP or the local database, the user will be redirected to Welcome.php. If the user credentials fail, they will return to login.php.

The start of this file includes connection.php for the connection to the local MariaDB database. The function sendExit() shortens the code length within the testing of the user credentials below. This function destroys any existing session and returns the user to login.php.

In this file there are several data validation and sanitation steps. The first are lines 6 and 7 which remove or replace html special characters to prevent invalid/unallowed sql queries. The next validation is the first if statement, testing the length of both the username and password. In the current setup of the credentials, neither the username or password are greater than 15 characters.

Following the initial if statement there are three options which include LDAP authentication, local database authentication, or returning to login.php (failing the first two options). The first test is if the username matches one that is included in the $user_array to see if the entered username is a valid “team” in the game. If so, a connection to the LDAP server is created and the password is tested with the authenticated username. If the authentication is successful, or the password matches with the username, a session is created with several session variables and the user is redirected to welcome.php, else the user is returned to login.php.

If the username does not match any of the available usernames in the $user_array, then the username credential is tested against the “moleary” username, which stems from the local database. Previously, all accounts were hosted and authenticated from the local database but currently only the “moleary” account is authenticated this way. If the username matches a hash of the password entered is created and an sql query follows. Using the connection.php file, the sql query is sent to the local database and returns the results. If there is an existing row that matches the username and password, then the user session is created with session variables and is sent to welcome.php.

All other requests to this file are sent back to login.php because they are not able to authenticate.

Connection.php

This file contains the variables and functions for connecting to a database to authenticate users and alter database tables. The function utilizes the four variables at the top of the file. Ensure that your php has the mysqli module installed and enabled for the connection to work properly.

Login.php

This file is the first file that users will see when they access their browser and search for the bank. This includes the title of the entire web app and a submission form with a username and password. This file transfers the user’s input to authentication.php for testing of the credentials entered.

Index.html

Carbon copy of login.php. If you don't provide a file to navigate to this is what you'll see if you just type the ip address.

Logout.php

This file is a helper file that is triggered by buttons throughout the webapp. This file redirects users to the login.php file after destroying their session. It also closes the connection to the database.

studentDashboard.php

This is the first file that any of the student teams will encounter once they've logged in with their creds as they are stored in the LDAP server. It displays website navigation buttons to another file goPhish.php and the logout.php page. The main purpose of this page is to display the corresponding team's submitted fishing attacks with a somewhat trimmed level of detail. That is to say that this page, as with many others, makes a query to the database, retrieves all of the necessary entries per the team's username and outputs them into an html table. Among the entries to this table is a details button that will allow the user to navigate to a separate details.php page that shows a full display of the information corresponding to the selected phishing attempt.

goPhish.php

This file houses the form responsible for collecting all of the information that is needed for students to submit their own phishing attack. This includes all of the text entries like the title, the hostname, the victim account's username and so on. It also houses the ability for users to submit either their own malware file or input a one-liner command to be executed (**the latter has yet to be implemented on the ansible side, but the input is still stored in the database**).

getPhish.php

The backend file responsible for processing the information sent by the form in goPhish.php. This takes all of the input as POST variables and creates a mysqli_query to insert them into the database. When accepting a file, a function is used to generate a UUID that is attached to the original filename for storage in the server. This UUID is also used later to create a corresponding log file once the malware is executed.

getPhish.js

Houses the javascript code that allows users to choose to upload either a file or a command. That's it.

stuDetails.php

In a similar fashion to the dashboard, all this file does is pull information from the database that corresponds to the id of the phishing attack that was selected. This page displays the full entry's worth of data in an easier to read Name/Texbox format as opposed to a table. If the file has been executed by the professor, the log file (ansible output) will also be displayed on this page.

profDashboard.php

Serves the exact same role as the student dashboard file with some added functionality. It still displays the trimmed down version of each phish entry for a team, but has buttons that allow the professor to display the phishing attempts of other teams without refreshing the page. The page will always initially display any attempts from 'france' because…it was at the top of the list. Clicking another team will refresh the table with the attempts of that team. These buttons are powered by jquery/ajax and a few accompanying files.

getTables.js

Checks the id of the button that was pressed and writes out that team's name as a string value, then passes it to a call to get_tables.php. The return of this call (a new table) will overwrite the contents of the <div> in profDashboard with the id 'getTables'.

Get_tables.php

Very much a 1 to 1 copy (or effectively so) of the code that's used in profDashboard.php to get the phish data from the database and write it out to a table, only this time using the string value that we get from getTables.js (same as the button pressed).

Details.php

As you might expect, this is largely the same file as stuDetails.php with a bit of added functionality. This includes the ability to change the values of the phishing attempt's grade, approval status, and add/edit comments. More importantly, this is also the page where the phishing attack can be executed by the professor by clicking the 'execute' button that fires execute.php.

changeStatus.php

In a similar fashion to getPhish, this just takes the contents of the form in details.php that contains all of the changeable information like grade/status/comments and updates the database.

Execute.php

The meat and potatoes of it all. This program rips whatever data it needs from the database to execute the ansible playbook command along with a few key pieces of data from the nagios server it retrieves through ssh. On top of executing the playbook, this program is also responsible for writing the log file that contains all of the ansible output associated with the attempt. As previously mentioned this file just uses the UUID of the malware to create the log file. Below all of that is a 1 to 1 copy of details.php that essentially returns the user to the previous page albeit with a new log file/ansible output.

profGrades.php

Quite similar to the professor dashboard, though with less detail and with the added condition that it only displays graded phishing attempts. It also has the ability to display the logs of all executed phishing attempts with a button press. The ability to refresh the table with per team buttons as in professor dashboard is retained.

getGrades.js

Basically another copy of getTables.js only this time the program returns a call to getGrades.php instead of get_tables.php

getGrades.php

A slightly adjusted version of get_tables.php to suit the table display format on profGrades.php.

Phish_pond.sql

The file responsible for building/rebuilding the database and creating the moleary user. This can be used to clear out all entries for a new semester and such, and is **the only file that should be in /home/ceo/Documents and NOT /var/www/html.**