**CST-407 Activity 2 Guide**

Contents

[Activity 2: Encrypted Data 2](#_Toc61930836)

[Part 1: Authentication and Encryption 2](#_Toc61930837)

[Part 2: Gnu PGP Tutorial 3](#_Toc61930838)

[Synchronous Encryption 4](#_Toc61930839)

[Asynchronous Encryption 7](#_Toc61930840)

[Part 3: Wireshark Network Packet Capture Demonstration 9](#_Toc61930841)

[Create a Login App 9](#_Toc61930842)

[Add Some Security to the App 12](#_Toc61930843)

[Packet Capturing with Wireshark 15](#_Toc61930844)

[Wireshark Instructions 15](#_Toc61930845)

[How to Use Wireshark on the Internet 22](#_Toc61930846)

[Exploring Web Application Protocols 22](#_Toc61930847)

# Activity 2: Encrypted Data

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission.

**Special Note: The starting code for select activities can be found in the "CST-407 Activity 2 Starting Code Zip File.” Refer to this when directed in the guide below.**

## Part 1: Authentication and Encryption

**Overview**

In this activity, students will examine authentication and encryption methods.

**Execution**

Execute this activity according to the following guidelines:

1. Review the associated resources located in the Topic Materials.
2. Research online resources for the phrases “How Does AES Encryption Work” and “Hashing vs. Encryption.”
3. Answer the following questions in complete sentences.
4. Make sure to properly reference and cite all examples and supporting evidence.
5. Make sure to utilize appropriate industry terminology.

**Identity and Authentication**

1. What is the difference between the terms *identify* and *authenticate*?
2. In what situations should each of these be used or not used?

**Authentication Methods**

Name three types of authentication other than a password. Explain the advantages and disadvantages of each.

**Caesar Encryption**

1. Give at least two examples of a type of encryption that is in the Caesar model.
2. Explain the process of deciphering a Caesar shift.

**AES and DES. Two Modern Encryption Standards**

1. How are AES and DES both different than the Caesar encryption model?
2. Explain the differences between AES and DES, and why one is superior to the other.
3. Explain how the XOR operator works and how it relates to encryption algorithms.
4. The AES algorithm uses as block cypher. What alternatives are there to block cypher? What are the advantages and disadvantages of a block cypher?

**Symmetric vs Asymmetric Encryption**

1. Explain the differences between symmetric and asymmetric encryption.
2. What are the advantages and disadvantages of each method?

**Private Key vs Public Key**

1. Explain the difference between a public and private key.
2. Research “RSA Algorithm Step by Step Example.” How do prime numbers relate to the RSA algorithm used in creating a public and private key?
3. Explain how quantum computing relates to cryptography.

**Hashing vs. Encryption**

1. Explain the purpose of a hashing algorithm. Explain how it is different than encryption.
2. Explain how a computer system can use hashing to ensure passwords are private.
3. Explain how hackers use a rainbow table and how you can defeat them.
4. Explain the impact that hashing and encryption have had in relation to historical, social, professional, ethical, and legal aspects of computing. To support your statement, provide at least one historical example of how encryption relates to ethics in computing.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. A Microsoft Word document containing an analysis of the questions proposed.

## Part 2: Gnu PGP Tutorial

**Overview**

In this activity, students will demonstrate both synchronous and asynchronous encryption, as well as encrypt email messages for secure communications. They will use the encryption tool GPG which is part of most Ubuntu distributions.

**PGP** is an acronym for Pretty Good Privacy, a computer program which provides cryptographic privacy and authentication.

**GnuPG** (aka **GPG**) is an acronym for GNU Privacy Guard, another computer program which provides cryptographic privacy and authentication. More information about the program is available on the official website located in the Topic Materials.

**Execution**

Execute this activity according to the following guidelines:

1. **Software Requirements:** VMWare. Ubuntu or Kali Linux Virtual Machine which should include GPG as part of the default instructions.

### Synchronous Encryption

1. Synchronous encryption, or “simple” encryption, uses the same key to lock and unlock a file.
2. First, familiarize yourself with the possible commands available to the utility. Open the linux terminal and type this command to get help.

**gpg –h**

1. We will use some of the commands that are listed in the help instructions.

*Encrypt a File Using Default Settings*

1. Open Linux.
2. Create a text document called **message.txt** and save it.



Figure 1. Plain Text Message created with a Text Editor

1. Run gpg to encrypt it.

**gpg -c message.txt**

1. The –c option is for symmetric encryption instead of the public key option. The public key option will be demonstrated in the next section of instructions.
2. This will create an encrypted file named **message.txt.gpg**. A text editor probably cannot display the contents properly. Try and open the encrypted file and you should see a binary file with gibberish characters.

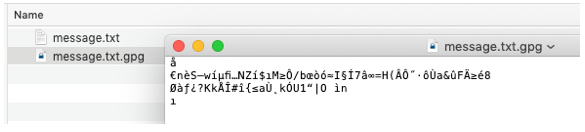


Figure 2. Message Encrypted in Binary Format

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

*Encrypt a File That can be Copied into an Email Message*

1. The armor option creates an output file that contains only ASCII standard characters instead of the default (and unreadable by a text editor) binary format.

**gpg -c --armor message.txt**

1. This will create an encrypted file named **message.txt.asc**. The **asc** stands for ASCII. You should be able to open it with a text editor.

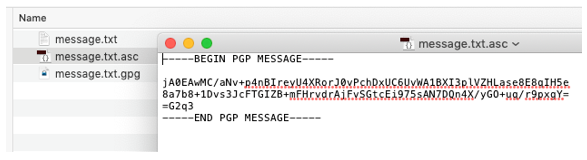


Figure 3. ASCII Version of the Encrypted Message

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

*Decrypt the Two Files*

1. Rename the **message.txt** file to **message-original.txt**

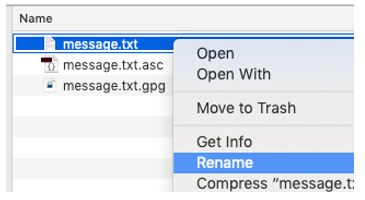


Figure 4. Rename the File

1. Decrypt the encoded text file. It should create message.txt again.

**gpg --decrypt message.txt.asc**

**gpg --decrypt message.txt.gpg**

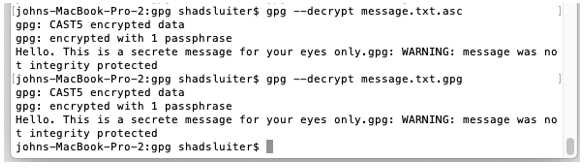


Figure 5. Message Decrypted to the Terminal

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the picture to explain what is being demonstrated.
2. Each of these commands will only display the message on the screen. To redirect the output back to a file, type in the commands as follows:

**gpg --decrypt message.txt.asc > message.txt**

**gpg --decrypt message.txt.gpg > message2.txt**

1. You should be able to open the decrypted message. Hopefully it has not changed, but can we prove that it has not changed?

*Verify That the File Has Not Changed with a Hash*

1. We will run a hashing function to get a digital signature on the file before and after the encryption process.

**md5sum message.txt**

**md5sum message2.txt**

1. This should generate a hash value unique to this text file.

**md5sum message-original.txt**

1. The hashes should match.

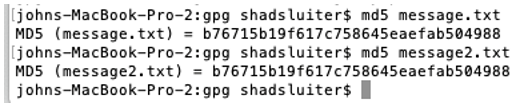


Figure 6. Matching Hash Values

1. Check the original file with a hash function.
2. Check the file that was decrypted with the same hash function. The two hashes should match to prove that the contents are the same.
3. You can use the “>” operator to dump console messages to a text file.
4. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

### Asynchronous Encryption

1. Create your own private and public key files.

**gpg --gen-key**

1. Follow the steps through the process:

**1 - RSA**

**2048 - key length**

**5y - expire date**

**shad - real name**

**shad.sluiter@gcu.edu - email**

**Always use encryption if your are paranoid - comment**

**o - OK**

**password - passphrase for your private key**

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

*Share your Key with Another User*

1. Use the –armor option to make it an ASCII file. Dump it to a file.

**gpg --export --armor youremail@email.com > shadsluiter-public-key.asc**

1. Share your public key in one or more of these ways.
   1. eMail. Copy the contents of the public-key.asc file into an email. Send it to your friend.
   2. Publish it to your personal webpage or social media app.
   3. Publish it on a key server where other people can copy it: <http://keyserver.ubuntu.com>, <https://pgp.mit.edu> or <https://keyserver.pgp.com>.

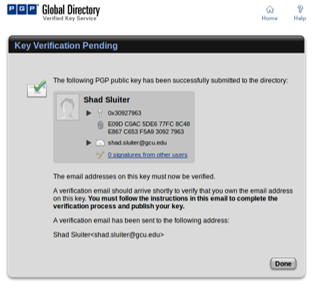


Figure 7. Online Directory of Public Keys

1. The fingerprint is a hash of your public key. You can use it to verify that a public key has not been tampered with.
2. Import a public key made by another person. This example shows how to import the file called **shad-gcu-pub-key.asc**.

**gpg -- import shad-gcu-pub-key.asc**

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

*How to View the Public Keys You Have Imported into Your Computer*

**gpg -- list-keys**

1. Your list of public keys will grow as you work with more individuals and share public keys.
2. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.
3. Encrypt a message in preparation for sharing it with another user. In this example, I am going to share a file (filename.txt) with my friend Bill Gates.

**gpg –-encrypt –-recipient --armor** [**bill.gates@microsoft.com**](mailto:bill.gates@microsoft.com) **filename.txt**

1. This will use the public key to encrypt the text file. There should be a new, encrypted file in the same directory as filename.txt. Type **ls** to confirm it is there.
2. Send the encoded message to your friend who will then decrypt it. You can copy the encrypted text into the body of an email message and safely send it via email, one of the most un-private forms of internet communication
3. Decrypt a message sent to you.
   1. You must first share your public key with a friend.
   2. Your friend imports the public key into GPG.
   3. He/she uses your public key to encrypt a message that is intended for you.
   4. He/she sends you an email with the ascii version of the encrypted file.
   5. You save the encrypted message to a file.
   6. You run the decrypt command to create an unencrypted text file.

**gpg –-decrypt message-from-friend.txt.asc > output.txt**

* 1. You must provide the password associated with your private key.

1. Take a screenshot at this point of the activity and paste it into a Word document. Place a caption below the image to explain what is being demonstrated.

**Note:** Make sure to utilize appropriate industry terminology when completing this activity.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Word document comprising all screenshots with captions.

## Part 3: Wireshark Network Packet Capture Demonstration

**Overview**

In this activity, students will become familiar with Wireshark. Application security is both an internal and external problem. The way in which an application sends data across a network will determine how secure the data remains. In this activity, you will see how Wireshark works. Wireshark is a popular hacking tool as well as network engineer’s diagnostic tool for viewing communication over a computer network.

1. **Tools Needed:** (a) Web server – Use MAMP, WAMP, etc. (b) Wireshark (c) Text editor
2. Using this tutorial, create a very simple application that has a login form and form processor. The application will run on a MAMP server using html and PHP code. We will create two versions of the app according to the instructions below. We will watch the network traffic using Wireshark and steal credentials from an unencrypted login session.

**Execution**

Execute this assignment according to the following guidelines:

### Create a Login App

**App Version 1 – No Security**

The starting code for this project can be found in the "CST-407 Activity 2 Starting Code Zip File," or you can follow the steps pictured below.

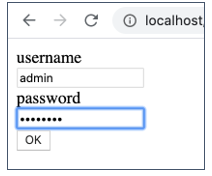


Figure 8. Login Screen

1. Open your text editor program. Create an html file and create a simple login form. You should see this file in the starting code.

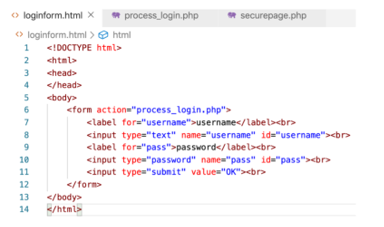


Figure 9. Login Form

1. Create a second file named **process\_login.php** to authenticate the user. The file contains three possible login credentials. In a real application, these credentials would be stored in a database instead of within the login controller code as shown here. This file is also found in the starter code.

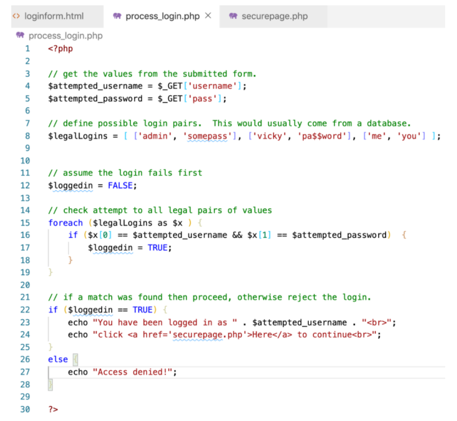


Figure 10. Login Process

1. Create the **securepage.php** file, which should only be displayed after the user logs in. In this version of the application, there is no check for authentication before displaying the page.

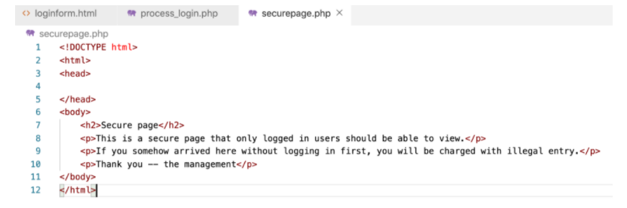


Figure 11. Secure Page

1. Notice there are several security flaws in this version of the app.
   1. The GET values from the login form are displayed in the browser’s URL in plain text. There should be some way to hide this. In the next version of the app, use the POST method instead of the GET method.
   2. There is no authentication check in the secured page. A user can simply type the URL in the browser without going through the login form. This will be fixed by adding a session variable to the app.
2. Take screen shots of the app showing that the login works, that the secure page is not really that secure and that the GET method shows the password in the browser’s URL address bar. Add a caption to each picture to explain what is being demonstrated.

### Add Some Security to the App

1. In the second version of this simple app, apply two fixes: (a) Change the form method from GET to POST and (b) use session variables to check for authentication before showing the secure page of the app.
2. Modify the html login file and add the POST method to the form.



Figure 12. Login Form

1. Modify the PHP **process\_login** page to respond to the login event.

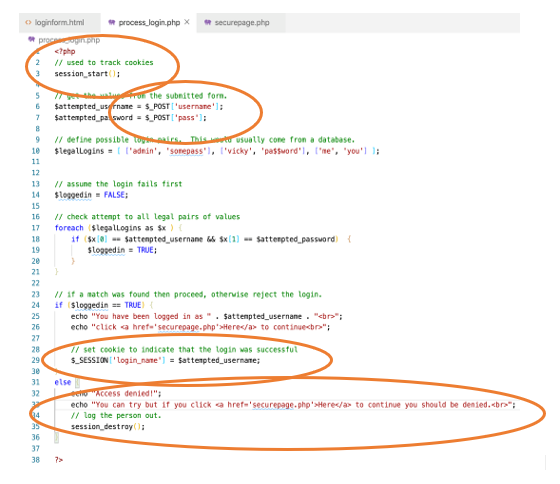


Figure 13. Login Form using Session Variables to Track User Login

1. Add **session\_start** in line 3. This allows the page to utilize session variables.
2. Change the expected parameters passed to this page to be sent in a POST request instead of GET (lines 6 and 7).
3. After a successful login, set a session variable to keep track of who just logged in (line 29).
4. If the login fails, destroy any session variables (line 35).
5. Modify the secure content page. It should first check to see if the proper session variable has been set. If not, then the page is not displayed.

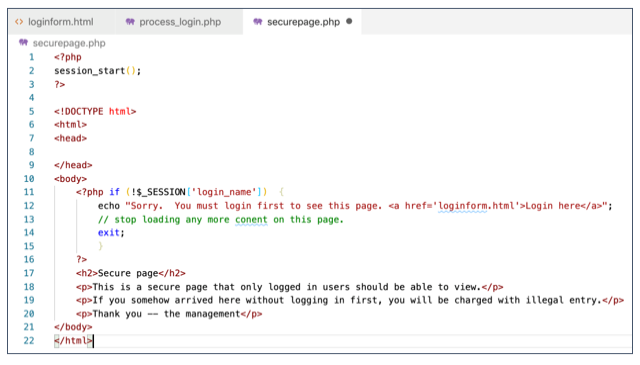


Figure 14. User Login Example

1. Line 2 initiates a session. Line 11 checks to see if a valid login session variable has been set by the process login page.
2. Run the program and attempt a login.
3. You should get either a success or fail depending on the password you provide. Notice that the URL does not contain the username and password anymore.

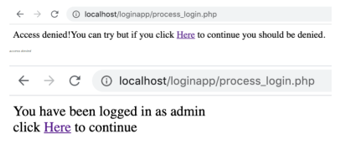


Figure 15. Login Failure

1. The secure page should be visible only if you successfully logged in.

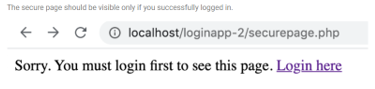


Figure 16. Login Success

1. Take screen shots showing the improvements in the program. Add a capture to each picture to explain what is being demonstrated.

### Packet Capturing with Wireshark

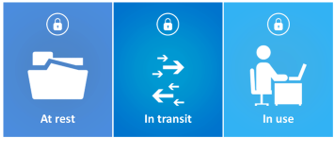
****

Figure 17. Three States of Data

About This Demonstration

In this section of the tutorial, capture the packets sent to and from the website and display them in plain text. This demonstration shows how easy it is to read unencrypted data being transferred over a network.

In data security, data is addressed in three phases: (1) At rest (2) in transit, and (3) in use. Each has its own issues regarding confidentiality, integrity, and availability. In the following steps, will focus on the second stage, the “in transit” phase of data. We will show that it is possible to compromise confidentiality if communications are not conducted in an encrypted channel.

### Wireshark Instructions

1. Be sure that the Wireshark application is installed. Refer to the instructions in the Course Materials.

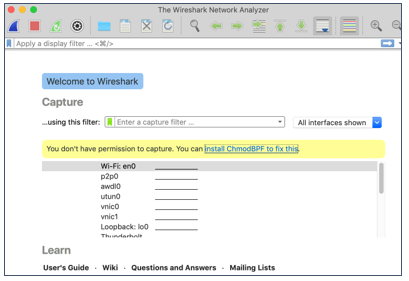


Figure 18. Wireshark Download

1. Launch the application.
2. Notice that you need to install **ChmodBPF** and relaunch the app.
3. Select the network card that you want to monitor. In this case, you want to monitor “**Loopback**” because the website we are looking at is **localhost**.

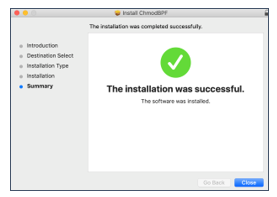


Figure 19. Wireshark Installation

1. If we wanted to capture traffic coming from Internet websites, we would select the WiFi (en0) card. Your computer likely has different network card names than mine.

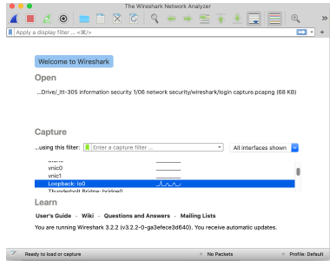


Figure 20. Choose the WiFi Card to Monitor

1. If the localhost web server is running, you will likely see some network traffic results.

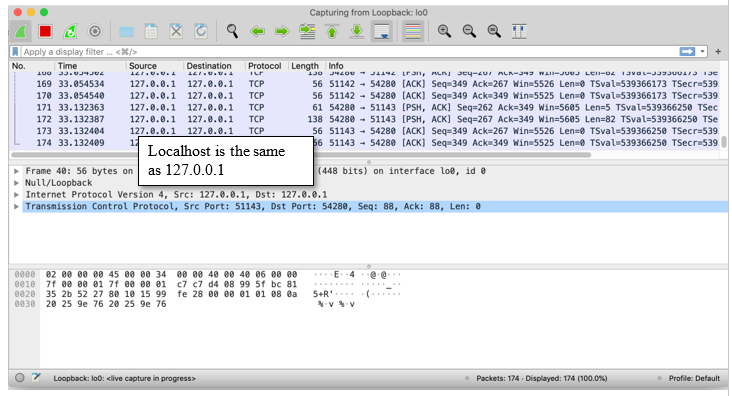


Figure 21. Localhost Traffic is Being Tracked

1. Click the shark fin to start capturing packets.



Figure 22. Start Button for Wireshark Tracking

1. While Wireshark is capturing packets, switch to the local server website and perform a login.

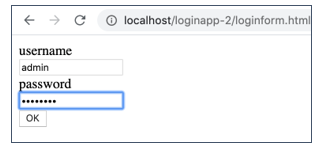


Figure 23. Local Server Login

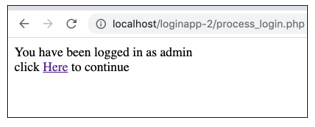


Figure 24. Successful Local Server Login

1. Return to Wireshark and stop capturing by clicking on the red stop button.



Figure 25. Wireshark Button to Stop Monitoring

1. You should see a long list of text. Each line in the report represents a packet of communications between your computer and the localhost web server.

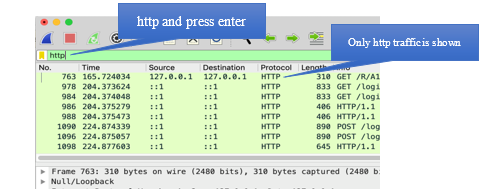


Figure 26. Filter Captured Packets to Show only HTTP Traffic

1. Apply a filter to show only “http” packets.
2. Select one of the GET packets. The packet on line 2 shows that the **loginform.html** page was loaded.

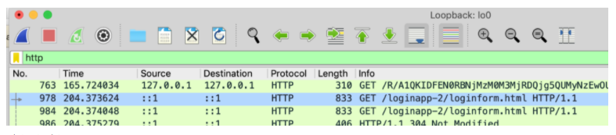


Figure 27. Selected the Loginform Event GET Packet

1. You can retrieve a more readable version of the packet by right-clicking the line in the log > Follow > HTTP Stream.

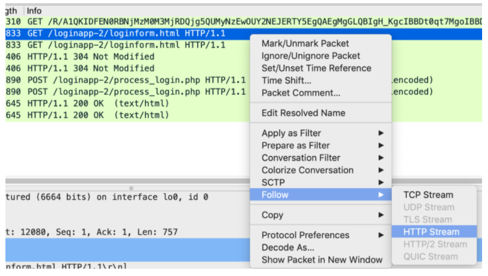


Figure 28. Show all Packets Associated with the Login Form’s GET Event

1. Take screen shots showing the packets that were captured in the login process. Add a caption to each picture to explain what is being demonstrated.
2. You can return to the filtered list by clicking the X in the filter line and re-entering the http filter.

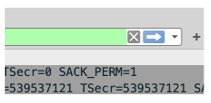


Figure 29. “X” Button to Clear the Packet Filter

1. Enter a more specific filter that only looks at POST events. Use **http.request.method == “POST”** as the filter.

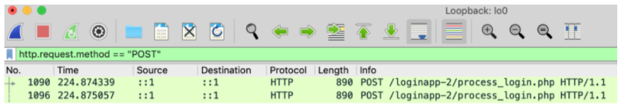


Figure 30. Filtering for POST Packets in Wireshark

1. Select one of the POST packets and expand the contents. You should be able to see the form items for username and password by opening the items in the bottom window.

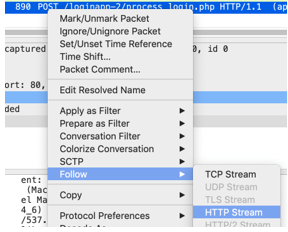


Figure 31. Showing all Associated Packets with the POST Event

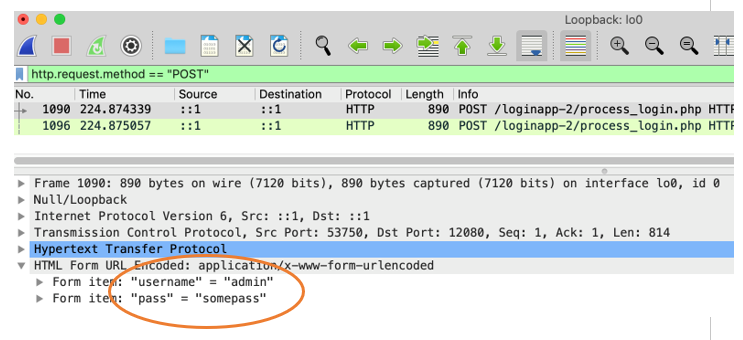


Figure 32. Expanded POST Packet Reveals Data Submitted via the Login Form

1. You can also display the entire packet in a more readable format by right-clicking > Follow > HTTP stream.
2. You can see the http traffic, which includes the cookie information about the form contents.

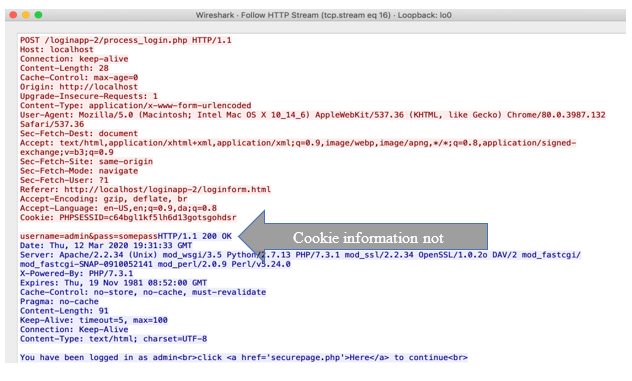


Figure 33. Cookie Information Showing Username and Password

1. Take screen shots showing packets captured during the login process. Add a caption to each picture to explain what is being demonstrated.

### How to Use Wireshark on the Internet

The demonstration we performed in this activity was conducted on a localhost website. In the real world, you would only need to change a few things to do the same task including the following:

1. **Wi-Fi NIC**. When starting Wireshark, select the Wi-Fi network card instead of Loopback. The traffic between your computer and one on the network must travel through the Wi-Fi card.
2. **Monitor Mode**. To capture packets from other computers in the area, set your network adapter to “Monitor Mode.” Any computer in a public café could potentially be monitoring your network traffic. Usually, the network card ignores any packets that are not indented for itself. If your network card supports monitor mode, then you can capture packets that travel between the Wi-Fi access point and your neighbor’s computers. Network cards in laptops frequently do not have a monitor mode. A USB Wi-Fi access card is usually required.
3. **Encryption.** Most websites today send all data in an encrypted format. Capturing packets from an https session are scrambled.

### Exploring Web Application Protocols

Some of the common protocols include not only HTTP, which are monitored in this exercise, but also those listed in Table 1.

Copy and paste the following table into a Word document. For each protocol, explain briefly what it is, its purpose, what type of application program is likely to utilize the protocol, and whether it is vulnerable to packet sniffers like Wireshark.

Make sure to utilize appropriate industry terminology.

**Table 1. Table of Common Internet Communication Protocols**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Protocol Initials** | **Full Name** | **Purpose** | **Type of Application Likely to Utilize the Protocol** | **Security Vulnerability to Wireshark** |
| **SMTP** |  |  |  |  |
| **IMAP** |  |  |  |  |
| **DNS** |  |  |  |  |
| **SSH** |  |  |  |  |
| **POP** |  |  |  |  |
| **FTP** |  |  |  |  |
| **TELNET** |  |  |  |  |

**Submission**

Submit the following to the learning management system:

**Part 1**

1. A Word document containing an analysis of the proposed questions.

**Part 2**

1. Word document comprising all screenshots with captions.

**Part 3**

1. Completed Table of Common Internet Communication Protocols and applicable screenshots with captions.