

# 4203 Final Project Report

Damus, Karl  
101196754

Yeager, Jack  
101180117

April 8, 2024

# Contents

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Brief History . . . . .	3
2.2	Purpose of the Study . . . . .	3
2.3	Hardware/Software Required . . . . .	4
<b>3</b>	<b>Literature Review</b>	<b>4</b>
3.1	What is FTP? . . . . .	4
3.2	What is SFTP? . . . . .	4
3.3	What is Packet Sniffing? . . . . .	5
3.4	Related Work . . . . .	7
3.4.1	Targeting FTP Servers to Compromise Protected Health Information . . . . .	7
3.4.2	A Personal Account . . . . .	8
<b>4</b>	<b>Comparison of FTP vs. SFTP</b>	<b>8</b>
<b>5</b>	<b>Attacks and Vulnerabilities on/in FTP</b>	<b>9</b>
5.1	Anonymous Authentication . . . . .	9
5.2	Brute Force Attack . . . . .	9
<b>6</b>	<b>Cryptographic Algorithms in SFTP</b>	<b>9</b>
6.1	Encryption Methods in SFTP . . . . .	10
<b>7</b>	<b>Our Implementation</b>	<b>11</b>
7.1	Tools and Software . . . . .	11
7.2	Design . . . . .	11
7.2.1	Architecture . . . . .	11
7.2.2	Security . . . . .	11
7.2.3	Client . . . . .	13
7.2.4	Logging . . . . .	14
7.3	Packet Sniffing . . . . .	15
7.4	Results . . . . .	15
7.5	Visualization of Our Tests . . . . .	17
<b>8</b>	<b>Conclusion</b>	<b>17</b>
<b>9</b>	<b>Appendix</b>	<b>19</b>
<b>10</b>	<b>References</b>	<b>19</b>

# 1 Abstract

We compare and contrast File Transfer Protocol (FTP) and Secure-FTP (SFTP) as well as discuss our own implementation of SFTP. Our implementation of SFTP, built on the shoulders of existing FTP technology, uses industry-standard cryptographic hashing algorithms to ensure secure file transfer over the network. This project outlines the design, security, and performance of securely sending files over a network.

First, we discuss the architecture and existing implementations of FTP and SFTP. We focus on the history and relevance of both technologies. We show that FTP is vulnerable to network attacks and how SFTP manages to be resilient towards attacks that are effective on FTP connections. We will briefly compare the two protocols.

Next, we will discuss some common attacks that FTP is vulnerable to, how SFTP secures these attacks, and what cryptographic algorithms SFTP employs to become secure.

# 2 Introduction

## 2.1 Brief History

Both FTP and SFTP are methods of transferring files over a network and have existed for many decades with FTP's introduction in 1971 and SFTP's introduction in 1997. FTP was introduced as a rudimentary file transfer protocol for ARPANET, the precursor to the Internet. Because of the small user base of ARPANET and its primary use for academic research, FTP was built with no security in mind. It is transferred entirely in plain text with no encryption meaning login information and file transfer data can easily be gathered with packet sniffers. Thus, means of encryption would have to happen independently of FTP (encrypt before being sent with a predetermined key between the sending and receiving party). Throughout its history, FTP has been updated several times with other industry innovations like its adoption of TCP/IP in 1980.

## 2.2 Purpose of the Study

In today's connected world, secure data transfer is of great importance. Packet sniffing, a technique used to capture and analyze packets sent over a network makes raw FTP transfers completely unsafe. To provide a thorough understanding of how vulnerable you are when using raw FTP, we deployed a custom FTP server on a Raspberry Pi and developed a simple web application to connect to the Pi and transfer files to the server.

This study examines the weaknesses and security implications of FTP and SFTP, and risks associated with packet sniffing when using FTP. We aim to

understand the intricacies of file transfers and the importance of making secure decisions when implementing a file transfer and/or storage service.

### **2.3 Hardware/Software Required**

For this project, we used a Raspberry Pi 3 Model B running Raspberry Pi OS Lite to run our FTP Server as well as our implementation of a secure FTP. To interface with the server, we built a simple web application using NodeJS that can upload files to the FTP server. With this all set in place, we are able to recreate attacks and make our observations on the protocols as well as implement our own encryption on top of the protocol.

## **3 Literature Review**

### **3.1 What is FTP?**

The FTP model depends on a FTP server with a filesystem that a user can log into to transfer files from the user's filesystem to the servers. Communication between the user and server and the file transfer occur over two respective TCP connections, the first for communication and signaling of commands from user to server (henceforth, Control Connection) and the second connection used for the data transfer itself (henceforth, Data Connection). Connection is first established by the user protocol interpreter communicating to server protocol interpreter to establish the Control Connection. It is through this connection that commands are sent to the server which dictates how and what files are being transmitted through the Data Connection.

Data Connection depends on default ports used for data transfer if ports are not specified by the user (default port for user side is same port used for Control Connection and default port for server side is the port adjacent to the port used for Control Connection). The user can specify a variety of means of data transfer and different types of data on top of the various commands including rudimentary commands used to manage the creation and deletion of file folders.[1]

### **3.2 What is SFTP?**

The SFTP protocol was introduced in 1997 and was developed as an extension to the SSH protocol and as a replacement of FTP. SFTP offered more security than FTP through means of encryption SSH offers while still maintaining the same functionality of FTP as means of accessing and controlling a remote filesystem. The addition of SSH to SFTP means the protocol is already significantly more secure than FTP with data being encrypted during transit as well as supporting authentication and all done with easier configuration than FTP.

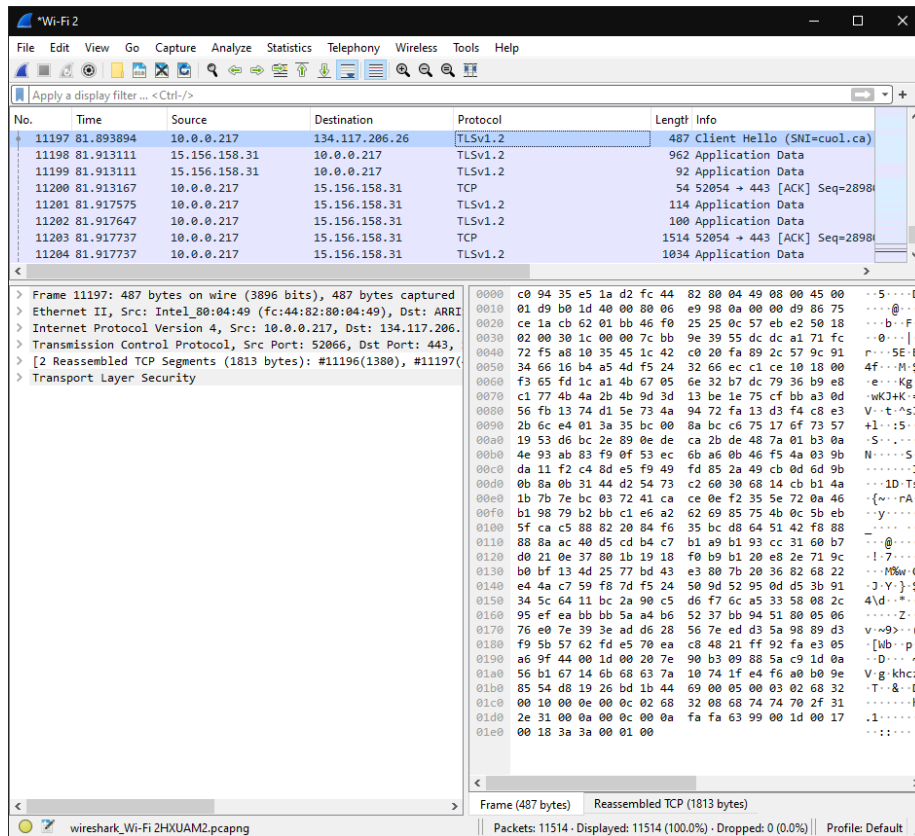
Because the protocol is built on SSH, much of the software for client and server comes as part of OpenSSH suite, as such SFTP servers rely on a configured SSH server and runs over the usual SSH port 22. Although FTP servers can be configured to implement SFTP as well as FTP. The standard client as well comes with OpenSSH and shares a lot of the same functionality as the FTP client. Packets of SFTP will all have fields for length, type and request-id, with additional fields depending on the type. For example, SSH\_FXP\_OPEN is defined by type 3 with the added fields for filename, desired-access, flags and attrs with flags being specification of operation (new file or existing etc) and attrs specifying attributes for the file.[2]

### 3.3 What is Packet Sniffing?

Packet sniffing is the process of capturing data sent over the local network. There are many tools available to capture packets over the network, the most commonly known being Wireshark. Packet sniffing is most often used for analyzing issues on a network however if applications are transferring insecure data (such as over FTP), the data transferred over the network can be viewed in its raw format. Thus, there is malicious capabilities with packet sniffing e.g., stealing usernames and passwords as well as intercepting confidential documents.

As an example, the following is a simple packet capture session of a connection to Carleton's Brightspace website from the local network. The conversation goes as follows:

- Client (10.0.0.217) sends a 'client hello' message to server (134.117.206.26) to initiate TLS handshake.
- Server acknowledges receipt of 'client hello' message.
- Server sends 'server hello, change cipher spec, encrypted handshake message' to specify TLS version, indicate transition to encrypted communication, and send encrypted data including server info and cryptographic parameters.
- Client sends same type of message to confirm the change and confirm the data sent by the server.
- Client sends encrypted application data to the server. The data contains encrypted HTTP/HTTPS requests and responses.
- Server acknowledges the receipt of application data.



Though we were able to see the packets in the conversation being sent, it is not possible to decrypt the data sent with any realistic computational power and time. To view the unencrypted data, one would have to set up and execute a man-in-the-middle attack. In this paper, we will display how data sent over FTP is not safe when packet sniffing is used.

However, it is important to note that anyone packet sniffing can observe what sites you are visiting. In the example below, I supplied the destination IP address to the website whois.com and discovered that the client was communicating with Carleton University.

```
NetRange:      134.117.0.0 - 134.117.255.255
CIDR:          134.117.0.0/16
NetName:       CARLETON1
NetHandle:     NET-134-117-0-0-1
Parent:        NET134 (NET-134-0-0-0-0)
NetType:       Direct Allocation
OriginAS:
Organization:  Carleton University (CARLET-1)
RegDate:       1989-05-25
Updated:       2021-12-14
Ref:           https://rdap.arin.net/registry/ip/134.117.0.0
```

```
OrgName:       Carleton University
OrgId:         CARLET-1
Address:       Computing and Communications Services
Address:       401 Administration Building
City:          Ottawa
StateProv:     ON
PostalCode:    K1S-5B6
Country:       CA
RegDate:       1989-05-25
Updated:       2013-05-17
Ref:           https://rdap.arin.net/registry/entity/CARLET-1
```

### 3.4 Related Work

#### 3.4.1 Targeting FTP Servers to Compromise Protected Health Information

In 2017, the Federal Bureau of Investigation (FBI) released a notice of criminals who specifically targeted FTP servers in use at medical and dental facilities. The adversaries were using anonymous mode to access the servers. The FBI mentions research from the University of Michigan which said over 1 million FTP servers were configured to allow anonymous access putting them at risk.[4] Though it has its uses, anonymous access mode can put your application at risk if not properly monitored. We will discuss the implications of allowing anonymous access on an FTP server in a later section.

### 3.4.2 A Personal Account

I have experience knowing about insecure FTP usage in a medical establishment. For personal reasons, I will refrain from mentioning the name of the place this relates to and will instead refer to it as 'The Medical Establishment'.

This occurred when I was in grade school. I would frequent The Medical Establishment after school. As an overly curious child, I constantly asked questions about how things worked. This included one of my longest fascinations, computers. I often was around the software being used by The Medical Establishment and naturally asked a lot of questions about their system. I came to learn that they were using outdated and insecure FTP for their medical records and were highly at risk because no one understood that it was worth the cost to update. I am not aware of any security breaches of The Medical Establishment but they were certainly at risk for many years. They have since upgraded their software.

– Karl Damus

## 4 Comparison of FTP vs. SFTP

While the development of FTP motivated the development of SFTP, there are many differences between the two protocols despite the similar functionality. The clearest difference and much of the reason for its existence is the improvements in security through encryption through SSH. Whereas FTP packets are sent in plaintext, SFTP packets are encrypted in transit adding protection against packet sniffing. Architecture wise, FTP utilizes two TCP connections for controlling user commands and transfer of files whereas SFTP utilizes one TCP connection for all of its functionality.

Another interesting difference is the history of development between the two protocols. It was mentioned before that FTP existed before the modern internet and had continuous updates throughout the internet's development through RFC memos, the same cannot be said with SFTP. Despite having variety of support for both client and server, SFTP only exists as internet drafts and never had a full RFC memo release. Along the development of SFTP, it was decided that SFTP is a type of file system protocol and not just simply a file access or transfer protocol which put it out of the purview of the working group at the time. It has not been picked up in a meaningful way since then and now only exists as drafts.



Feature	FTP	SFTP
Protocol	TCP	TCP
Encryption	None	SSH-Based
Authentication	Username and Password	Same as FTP + key-based authentication
Data Integrity	None	Yes
Data Transferred	Plain text	Encrypted

## 5 Attacks and Vulnerabilities on/in FTP

### 5.1 Anonymous Authentication

Anonymous authentication mode allows a user to access the FTP server without any credentials. When you execute ‘ftp [IP-ADDRESS]’ and are prompted for a username you can enter ‘anonymous’ and any password and will be loaded into the directory specified by the administrator of the FTP server.

It is possible that the directory you have access to already has sensitive files, this would be just inviting someone to steal your private information. If the anonymous user has write permissions, an adversary could modify files and/or create malicious files to attack legitimate users and get their personal information. With write permissions, an adversary could also use your server as their own personal spot to store illegal and even nefarious data and use your server as a distribution point.

### 5.2 Brute Force Attack

Brute-forcing is quite common and preys on users who do not use secure passwords. Tools such as Hydra on Kali Linux provide the ability to execute brute-force attacks on more than 50 different protocols including FTP. Hydra is incredibly easy to install and use which makes choosing a secure password of utmost importance.

## 6 Cryptographic Algorithms in SFTP

SFTP’s improvement over FTP is through its encryption through SSH. SSH provides not only encryption but authentication and integrity as well to the file transfer service. On top of the use of passwords, authentication is secured between client and server through means of RSA public key cryptography. It begins with the client initiating a connection to the SFTP server with the SFTP server responding to the client with its public key and any other required information. Key exchange then takes place between the client and server by both sending a ‘SSH\_MSG\_KEX\_INIT’ message which uses the server and client’s private and public keys and RSA to generate a session key which encrypts the session between client and server.

The transmission between server and client is encrypted using the session key and a symmetric encryption scheme. One common scheme used, and the one we will be using is the AES encryption scheme. AES is a block cipher scheme meaning it encrypts the plaintext through breaking it up into blocks of typically 128 bits. It encrypts these various blocks using round keys which is generated off of the supplied session key. As well AES performs three different invertible transformations on the blocks; sub-bytes which applies a substitution table on each byte, shifts rows according to an offset and mix columns operation. The block after having gone through the transformations is then encrypted with the corresponding byte of the round key. Decryption involves doing the inverse of the same steps.[7]

As a final step, SSH provides data integrity of transmissions sent and received through the use of SHA-256 hashing and hashed message authentication codes (HMAC).

## 6.1 Encryption Methods in SFTP

There are two methods of encryption that SFTP can use. Symmetric and asymmetric. Briefly reviewing the differences, symmetric key encryption uses the same key to encrypt and decrypt while asymmetric key encryption uses separate keys for encryption and decryption. The following is a detailed list of the differences:

Symmetric Key Encryption	Asymmetric Key Encryption
Fast	Slower
Ease of use	Complex implementation
Larger data sizes	Smaller data sizes
Less secure	More secure

Symmetric key encryption is most commonly used in SFTP but is less secure than SFTP using asymmetric key encryption. SFTP also implements a cryptographic hashing algorithm to increase security and ensure data integrity during its transfer from one place to another. In addition to the method listed above, there are the following encryption algorithms that are available to use in SFTP:

- AES
- ARCFOUR
- CBC
- CAST128

- 3DES
- Blowfish
- Twofish

## 7 Our Implementation

We will discuss the tools and software, design, setup, implementation, and testing process as well as the results of our custom implementation of SFTP and our packet sniffing.

### 7.1 Tools and Software

- Raspberry Pi 3B
- Raspberry Pi OS
- vsftpd-3.0.5
- Node.js
- crypto
- basic-ftp

### 7.2 Design

#### 7.2.1 Architecture

A simple client server model design where the client initiates connection, uploading, and downloading files from the server.

- Use Transmission Control Protocol (TCP) to connect to the server.
- Design client site to upload, view, and download files from the server.
- Design the Node routes for passing data between the client and the server.

#### 7.2.2 Security

Implement AES-256 encryption scheme on the client side using crypto module from npm.

```

1 // generate encryption key for AES-256 (length 32 bytes)
2 const generateEncryptionKey = (length = 32) => {
3   return crypto.randomBytes(length);
4 };
5
6
```

```

7 // encrypt a single file
8 const encryptFile = (inputPath, outputPath, key) => {
9   const iv = crypto.randomBytes(16);
10  const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);
11  const input = fs.createReadStream(inputPath);
12  const output = fs.createWriteStream(outputPath);
13
14  input.pipe(cipher).pipe(output);
15
16  cipher.on('error', (err) => {
17    console.log('Encryption Error: ', err);
18  });
19
20  input.on('error', (err) => {
21    console.log('Input Error: ', err);
22  });
23
24  output.on('error', (err) => {
25    console.log('Output Error: ', err);
26  });
27 }
28
29 // decrypt a single file
30 const decryptFile = (inputPath, outputPath, key) => {
31   const input = fs.createReadStream(inputPath);
32   const output = fs.createWriteStream(outputPath);
33   let iv;
34
35   input.once('readable', () => {
36     const chunk = input.read(16);
37     iv = chunk;
38   });
39
40   input.on('end', () => {
41     const decipher = crypto.createDecipheriv('aes-256-cbc', key
42       , iv);
43
44     input.pipe(decipher).pipe(output);
45
46     decipher.on('error', (err) => {
47       console.log('Decryption Error: ', err);
48     });
49
50     input.on('error', (err) => {
51       console.log('Input Error: ', err);
52     });
53
54     output.on('error', (err) => {
55       console.log('Output Error: ', err);
56     });
57   });
58 };

```

### 7.2.3 Client

Implement uploading and viewing files client interaction, retrieval, and uploading.

```
1 <form action="/upload" method="post" enctype="multipart/form-data">
2   <input type="file" name="file">
3   <button type="submit" name="submit">UPLOAD</button>
4 </form>
5
6 <form action="/files" method="get">
7   <button type="submit" name="submit">View Files</button>
8 </form>
```

---

```
1 app.get('/files', async (req, res) => {
2   const client = new ftp.Client();
3
4   try {
5     await client.access(clientOptions);
6     const files = await client.list("files");
7     await client.close();
8
9     res.send(files);
10  } catch(err) {
11    console.log('FTP Error: ', err);
12    res.status(500).send('Error fetching files from FTP');
13  }
14 });
15
16 app.post('/upload', upload.single('file'), async (req, res) => {
17   if (!req.file) {
18     return res.status(400).send('No file uploaded.');
```

```
19   }
20
21   const fileName = req.file.filename;
22   const filePath = req.file.path;
23   const encryptedFilePath = path.join('encrypted', fileName);
24
25   const key = generateEncryptionKey();
26
27   encryptFile(filePath, encryptedFilePath, key);
28
29   try {
30     const client = new ftp.Client();
31     await client.access(clientOptions);
32     await client.uploadFrom(encryptedFilePath, '/files/' + fileName
33   );
34     await client.close();
35
36     res.send('File uploaded to ftp: ${fileName} at ${filePath}');
37   } catch(err) {
38     console.log('FTP Error: ', err);
39     res.status(500).send('Error uploading file to FTP');
40   }
41 });
```

### 7.2.4 Logging

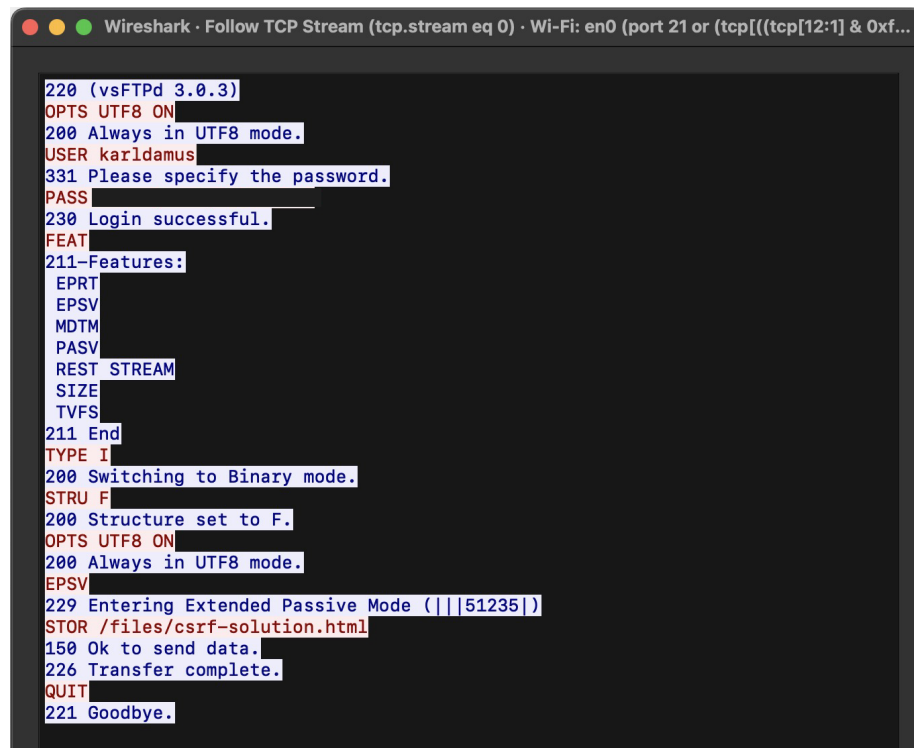
As an added security measure, to keep a constant eye on activity on the server, I created a simple script that will send me all the uploaded files within the past hour-ish. This occurs when a file is uploaded if the last time a log was sent was over an hour.

```
1 app.post('/upload', upload.single('file'), async (req, res) => {
2   . . .
3
4   const currentTime = new Date();
5
6   const fileName = req.file.filename;
7   const filePath = req.file.path;
8   const encryptedFilePath = path.join('encrypted', fileName);
9
10  // check if currentTime is 1 hour past lastLogTime
11  if (currentTime - lastLogTime > 3600000) {
12    // send email
13    message = "";
14    for (let i = 0; i < uploadedFiles.length; i++) {
15      message += uploadedFiles[i] + "\n";
16    }
17
18    sendMail(message);
19
20    uploadedFiles = [];
21    lastLogTime = currentTime;
22
23  } else {
24    // add details to uploadedFiles array
25    uploadedFiles.push("File: " + fileName + " at " + filePath
26      + " uploaded at " + currentTime);
27
28    . . .
29  });
30
31  function sendMail(message) {
32    var mailOptions = {
33      from: myEmail,
34      to: myEmail,
35      subject: 'Uploads from the past hour',
36      text: message
37    }
38
39    transporter.sendMail(mailOptions, function(error, info) {
40      if (error) {
41        console.log(error);
42      } else {
43        console.log('Email sent: ' + info.response);
44      }
45    });
46  }
```

### 7.3 Packet Sniffing

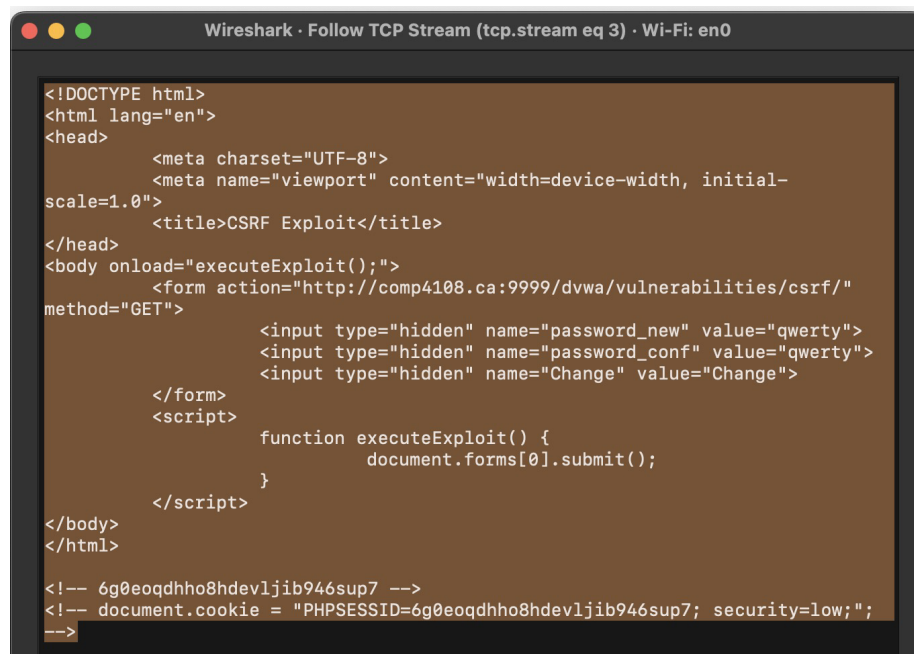
In order to test the security of our FTP server and then our custom SFTP implementation built on the shoulders of FTP we used Wireshark to capture transfers between the client and Raspberry Pi server. I was able to filter Wireshark to only show FTP data by setting the display filter to `ftp`. There are two cases we tested. One with raw FTP and one with our encryption added. I managed to capture the packets and view them in both cases however when encryption was added the packets were encrypted and not readable or able to be decrypted without the key used.

### 7.4 Results

A screenshot of the Wireshark network protocol analyzer. The top status bar shows 'Wireshark · Follow TCP Stream (tcp.stream eq 0) · Wi-Fi: en0 (port 21 or (tcp[12:1] & 0xf...'. The main pane displays the raw text of an FTP session. The text is color-coded: red for client commands, blue for server responses, and black for status messages. The session starts with a client greeting '220 (vsFTPD 3.0.3)', followed by server options 'OPTS UTF8 ON' and '200 Always in UTF8 mode.'. The client sends 'USER karldamus' and the server responds '331 Please specify the password.'. The client sends 'PASS' and the server responds '230 Login successful.'. The client sends 'FEAT' and the server lists features '211-Features: EPRT, EPSV, MDTM, PASV, REST STREAM, SIZE, TVFS'. The client sends '211 End', 'TYPE I', and the server responds '200 Switching to Binary mode.'. The client sends 'STRU F' and the server responds '200 Structure set to F.'. The client sends 'OPTS UTF8 ON' and '200 Always in UTF8 mode.'. The client sends 'EPSV' and the server responds '229 Entering Extended Passive Mode (||51235|)'. The client sends 'STOR /files/csrf-solution.html' and the server responds '150 Ok to send data.'. The client sends 'QUIT' and the server responds '226 Transfer complete.'. The session ends with '221 Goodbye.'.

```
220 (vsFTPD 3.0.3)
OPTS UTF8 ON
200 Always in UTF8 mode.
USER karldamus
331 Please specify the password.
PASS
230 Login successful.
FEAT
211-Features:
EPRT
EPSV
MDTM
PASV
REST STREAM
SIZE
TVFS
211 End
TYPE I
200 Switching to Binary mode.
STRU F
200 Structure set to F.
OPTS UTF8 ON
200 Always in UTF8 mode.
EPSV
229 Entering Extended Passive Mode (||51235|)
STOR /files/csrf-solution.html
150 Ok to send data.
226 Transfer complete.
QUIT
221 Goodbye.
```

Above is the captured conversation between the client and server to log into the server and then view files stored in the root directory. This conversation is completely unencrypted and therefore viewable to anyone capturing the packets on the network.

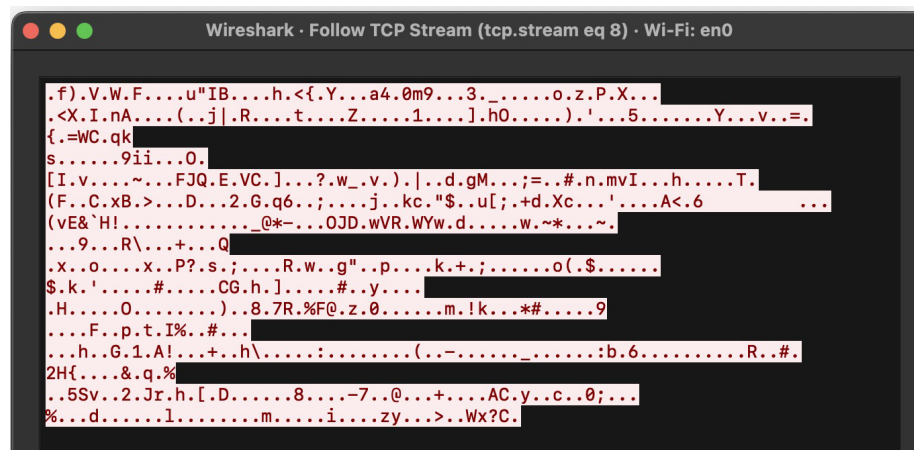


Wireshark · Follow TCP Stream (tcp.stream eq 3) · Wi-Fi: en0

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>CSRF Exploit</title>
</head>
<body onload="executeExploit();">
  <form action="http://comp4108.ca:9999/dvwa/vulnerabilities/csrf/"
method="GET">
    <input type="hidden" name="password_new" value="qwerty">
    <input type="hidden" name="password_conf" value="qwerty">
    <input type="hidden" name="Change" value="Change">
  </form>
  <script>
    function executeExploit() {
      document.forms[0].submit();
    }
  </script>
</body>
</html>

<!-- 6g0eoqdhho8hdevljib946sup7 -->
<!-- document.cookie = "PHPSESSID=6g0eoqdhho8hdevljib946sup7; security=low;";
-->
```

We were then able to view the contents of the file that was transferred. This is pictured above, captured entirely in Wireshark.



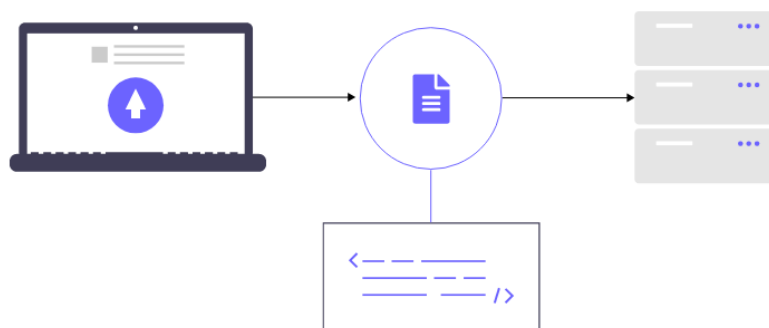
Wireshark · Follow TCP Stream (tcp.stream eq 8) · Wi-Fi: en0

```
.f).V.W.F....u"IB....h.<{.Y...a4.0m9...3.....o.z.P.X...
.<X.I.nA....(..j|.R....t....Z.....1....].h0.....).'.5.....Y...v...=.
{.=WC.qk
s.....9ii...0.
[I.v....~...FJQ.E.VC.]...?.w_.v.).|.d.gM...;=..#.n.mvI...h.....T.
(F..C.xB.>...D...2.G.q6...;...j..kc."$.u[;.+d.Xc...'....A<.6
(vE&`H!....._@*-...OJD.wVR.WYw.d.....w.~*...~.
...9...R\...+...Q
.x.o....x..P?.s.;...R.w..g"...p....k.+;.....o(.$.....
$.k.'.....#.....CG.h.].....#.y....
.H.....0.....).8.7R.%F@.z.0.....m.!k...*#.....9
...F..p.t.I%..#...
...h..G.1.A!...+..h\.....:.....(.-....._.....:b.6.....R..#.
2H{...&.q.%
..5Sv..2.Jr.h.[.D.....8.....-7..@...+...AC.y..c..0;...
%.d.....l.....m.....i.....zy...>..Wx?C.
```

After enabling our custom encryption, this was the result of the same captured conversation. The file was not viewable.



## 7.5 Visualization of Our Tests



Above is an example of an attacker intercepting the file being transferred with no security implementations, just plain FTP. The attacker is able to view the raw contents of the file.



Above is an example of an attacker intercepting the file being transferred with encryption enabled. The attacker is only able to view the encrypted conversation between the client and the server.

## 8 Conclusion

File transfer protocols are some of the most rudimentary protocols but they are equally as well the most necessary in order for a network to have any meaningful function. It is for that purpose we chose to study file transfer protocols that have been used historically and their security. First beginning our discussion

with FTP, we analyzed both its history and architecture, showcasing FTP's lack of any encryption. Analyzing various attacks and showing how it's vulnerable to simple packet sniffing eavesdropping attacks before moving the discussion to SFTP and how its encryption through SSH circumvents a lot of these attacks. We then discuss our implementation of an FTP server using a Raspberry Pi with added encryption and a web client to interface with the server to show how an implementation of SFTP looks and how it prevents some of the attacks.

## 9 Appendix

### Work by Karl Damus:

- Implementation of the web application including encrypting data, executing packet sniffing, and analyzing the data
- Cryptographic algorithms research
- Sections and Sub-sections: 2.2, 3.3, 3.4, 5, 6.1, 7

### Work by Jack Yeager:

- FTP/SFTP protocols research
- FTP implementation on Raspberry Pi
- Sections and Sub-sections: 1, 2.1, 2.3, 3.1, 3.2, 4, 6, 8

## 10 References

- [1] Postel, J. “RFC 959 - File Transfer Protocol.” IETF Datatracker, Oct. 1985.
- [2] Galbraith, Joseph and Saarenmaa Oskari. “IETF WIP- SSH File Transfer Protocol.” IETF Datatracker, Oct. 2001.
- [3] “22 - Pentesting SSH/SFTP: HackTricks.” HackTricks, [book.hacktricks.xyz/network-services-pentesting/pentesting-ssh](https://book.hacktricks.xyz/network-services-pentesting/pentesting-ssh).
- [4] “Cyber Criminals Targeting FTP Servers to Compromise Protected Health Information.” FBI, [info.publicintelligence.net/FBI-PHI-FTP.pdf](https://info.publicintelligence.net/FBI-PHI-FTP.pdf).
- [5] Hannah, Adrian. “Packet Sniffing Basics.” ACM Digital Library.
- [6] Spencer, Patrick. “Mastering SFTP Security: A Comprehensive Guide on Implementing Encryption.” Kiteworks, 2023, [kiteworks.com/secure-file-transfer/sftp-encryption/](https://kiteworks.com/secure-file-transfer/sftp-encryption/).
- [7] National Institute of Standards and Technology (2001) Advanced Encryption Standard (AES) (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publications (FIPS) 197. [nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf)