Knockoff Drive: A Portable Data Storage System

Nicholas Clark, Thomas Evangelista, John Scarfo, Garrett Stonis, Chen-Hsiang Yu and Mira Yun
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{clarkn3, evangelistat1, scarfoj, stonisg, yuj6, yunm}@wit.edu

***Abstract* – Network-based data storage is popular, but the privacy and portability involved in using such services raise a serious concern to the public nowadays. To remedy these issues, we propose to design and create a private and portable storage solution on a mobile device. Due to the fact that micro computers are more portable and affordable, we chose Raspberry Pi as an example to demonstrate the idea. *Knockoff Drive* is a system that provides private and portable data storage service for the users. The system consists of a HTTP server, FTP server and Web interface for data access. It is not only an affordable and alternative solution to file transferring on the go, but it also increases data security and privacy.**

Keywords *– HTTP server; FTP server; Web interface; Raspberry Pi*

I. INTRODUCTION

In the U.S., the Internet penetration has reached 84.2% and 39% of online adults use online cloud storage services [1, 2]. However, security concerns and risks of using this kind of services were raised, either enterprises or individual users [3, 4].

Instead of compromising the safety in data, we came up with an idea to bypass these third-party servers and create a user-oriented solution that might be easy-to-carry for an individual, i.e. fit into the palm of the user's hand. Because of the advance of Internet of Things (IoT) research and applications, the cost of computing devices has reached to an affordable price, we believe it is an opportunity to use this kind of computing device as a personalized replacement for existing internet services.

Based on above consideration, we propose to use a tiny, single-board computer to create a portable data storage server. The main research question we want to address is: how do we design a portable data storage server for an individual user? The design must be affordable for the public and customizable for different users.

In this paper, we present a primitive result of this experimental learning, named *Knockoff Drive*, which is a secure, portable data storage server built with a small, single-board computer, named Raspberry Pi. The system not only contains a Web server to provide web interface for data access, but it also has a FTP server to support upload and download service for the data. When a file is chosen to upload or download, the Web server will communicate with the FTP server to place the file in a special folder on a mobile device. To demonstrate the idea, we chose Raspberry Pi 3 model B as an experimental device.

In the following, we will refine the research question, present the current design and summarize our findings. At the end of the paper, we will conclude our work with some suggestions.

II. PROBLEM AND PROPOSED IDEA

Security breaches and risks of using third-party services raise a concern in today's computing world. In addition, there is a lack of a free, personalized, unlimited cloud storage system. The main research question is: What is the solution that not only addresses security concern, but it also provides a free, personized and unlimited cloud storage?

We propose to combine an FTP server with a Web server on a personal mobile device. *Knockoff Drive* is the system we propose to demonstrate the idea, which uses Raspberry Pi 3 model B as a personal mobile device to host an FTP and HTTP servers. The system can be used to download files from the Raspberry Pi to the computer requesting the file. The system can be expanded to hold multiple users on a single Raspberry Pi and have the files in a single folder. As for the storage, since MicroSD and external hard drives are becoming cheaper, *Knockoff Drive* can expand the internal or external storage easily depending on the amount of data they need. This capability removes the need to rely on a third-party storage system and allows a personalized alternative for customization and reliability.

III. KNOCKOFF DRIVE

The system consists of two parts: Web Service, FTP Service, HTTP Server and a single-board computer. In this section, we would like to explain the detailed design and implementation of the system.

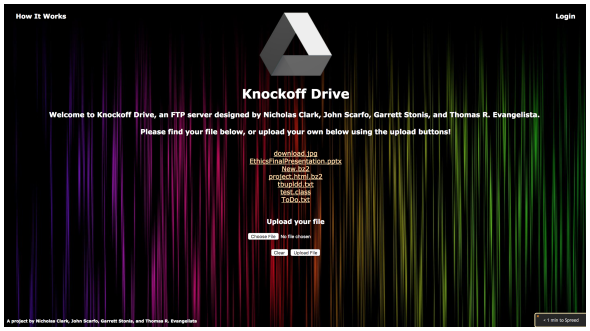

Figure 1. The Web site for access on the mobile device

A. *Web Server and UI Interface*

To begin with, we started from creating a web site in the mobile device to provide interaction for the users. The website is implemented by using HTML 5, CSS 3 and JavaScript as shown in Figure 1.

HTML is integrated first and is used to provide the content of the webpage. Unlike traditional programming languages, HTML is a markup language and uses tags to add titles, headings, images, and links to websites. For instance, creating a paragraph requires the use of a "p tag" or the letter "p" enclosed in arrowed brackets, which looks like this <p> [1]. These elements are all enclosed further inside of a <body> tag, which is the actual content the user will see and interact with. In the title tags we link the CSS for the page. The HTML code used for the index page of *Knockoff Drive* can be seen below in Figure 2.

```
<!DOCTYPE HTML>

<html>

<head>
    <title>Knockoff Drive - Upload/Download</title>
    <link href="index.css" type="text/css" rel="stylesheet" />
</head>

<body>

    <br />

    <a href="index.html"><img src="logo-drive2.png"></a>

    <a href="login.html"><h4 id="login">Login</h4></a>
    <a href="howitworks.html"><h4 id="howitworks">How It Works</h4></a>

    <h1>Knockoff Drive</h1>

    <h2>Welcome to Knockoff Drive, a free/open-source FTP server. I love you!</h2>

    <br />

    <form action="upload.php" method="post" enctype="multipart/form-data">
        <p><b>Upload your file</b></p>
        <input type="file" name="fileToUpload" id="fileToUpload" /><br />
        <br />
        <a href="index.html"><button type="button" id="button1">Clear</button></a>
        <input type="submit" value="Upload File" name="submit" />
    </form>

    <br />
```

Figure 2. HTML for the index page of *Knockoff Drive*

CSS is used to style the webpage. This is performed by selecting HTML tags and applying different colors and effects to them. For example, paragraphs can have their fonts, font color, backgrounds, or size changed. For *Knockoff Drive*, we attempted to ensure readability and simplicity. This was achieved by limiting the amount of content per page and using a simple background with white text. Basic syntax is as follows: Selector: {property: value} [2].

Once the basic outline of the website was created, we use JavaScript to add functionality to the webpage. Luckily, *Knockoff Drive* did not require much JavaScript to be written so we could focus on the PHP needed to upload a file. Unlike HTML and CSS, JavaScript is a fully-fledged programming language capable of event-driven programming with texts, arrays, and dates [3]. For *Knockoff Drive*, we created a basic function to allow users to login to the index of the website that have access to the right username and password.

```
<?php
$ftp_server = "169.254.67.248";
$ftp_user_name = "Temporary";
$ftp_user_pass = "Thia&Ugg";
$destination_file = "/home/temp";
$source_file = $_FILES['uploadedfile']['tmp_name'];

// set up basic connection
$conn_id = ftp_connect($ftp_server, 21);
ftp_pasv($conn_id, true);

// login with username and password
$login_result = ftp_login($conn_id, $ftp_user_name, $ftp_user_pass);

// check connection
if ((!$conn_id) || (!$login_result)) {
    echo "FTP connection has failed!";
    echo "Attempted to connect to $ftp_server for user $ftp_user_name";
    exit;
} else {
    echo "Connected to $ftp_server, for user $ftp_user_name";
}

// upload the file
$upload = ftp_put($conn_id, $destination_file, $source_file, FTP_BINARY);

// check upload status
if (!$upload) {
    echo "FTP upload has failed!";
} else {
    echo "Uploaded $source_file to $ftp_server as $destination_file";
}
// close the FTP stream
ftp_close($conn_id);
?>
```

Figure 3. Upload data to the FTP server via a PHP script

Lastly, the PHP code was needed to post the selected file to the FTP server. PHP is a server scripting language that has its scripts executed on the server. For *Knockoff Drive*, we used PHP to set up a connection and login to the FTP server from the Web server. Then, after connecting the servers, we needed to make sure both connections were made through use of an error check. After the check has been completed, the upload process begins using the "ftp_put" function and it takes the source file and uses "FTP_BINARY" file encoding type

to transfer the file to a destination folder [5] as shown in Figure 3. After one final error check is made to make sure the file has been uploaded, the connection to the server is closed.

B. *FTP Server*

One key feature of *Knockoff Drive* is to upload files to the storage. To provide this feature, we set up an FTP server on the mobile device, i.e. Raspberry Pi, which runs Raspbian Linux on it. We used ProFTPD, which is a secure FTP server program that operates in the server side to host an FTP server on port 21 [6]. The software takes in its configuration settings from a file created in the */etc/proftpd* directory, named *proftpd.conf*. [7].

```
pi@raspberrypi:~ $ sudo useradd -d /Desktop/ftp -e 2017-12-12 temporary
pi@raspberrypi:~ $ sudo passwd temporary
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Figure 4. The Linux commands to create a new user

We started by creating a new user on the Raspberry Pi, named "temporary" with the password "Thia&Ugg" that expires on a certain time [8]. The process is illustrated as Figure 4. After the creation of a new user account for the Pi, the new account was added into the FTP's *proftpd.conf* file to allow the user to access the server. We allowed the new user "temporary" access to his own home directory (*/home/temp*) but nowhere else, maximizing the security of our FTP server [7].

```
[C02S24V6G8WM:Desktop ThomasEvangelista$ ftp 169.254.67.248
Connected to 169.254.67.248.
220 ProFTPD 1.3.5 Server (169.254.67.248) [::ffff:169.254.67.248]
Name (169.254.67.248:ThomasEvangelista): temporary
331 Password required for temporary
Password:
230 User temporary logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||62975|)
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 temporary temporary        0 Nov 29 01:24 amazingFile.bz2
-rwxrwxrwx   1 0        0            13204 Nov 28 20:03 download.jpg
-rw-r--r--   1 temporary temporary  8785767 Nov 28 23:21 EthicsFinalPresentation.pptx
-rwxrwxrwx   1 0        0                0 Nov 28 21:07 New.bz2
-rwxrwxrwx   1 0        0              525 Nov 28 20:03 project.html.bz2
-rw-r--r--   1 temporary temporary     1171 Nov 28 22:10 tbupldd.txt
-rwxrwxrwx   1 0        0              406 Nov 28 20:03 test.class
-rw-r--r--   1 temporary temporary     1171 Nov 28 21:05 ToDo.txt
226 Transfer complete
ftp> get New.bz2
local: New.bz2 remote: New.bz2
229 Entering Extended Passive Mode (|||55294|)
150 Opening BINARY mode data connection for New.bz2
     0        0.00 KiB/s
226 Transfer complete
ftp> put temp.java
local: temp.java remote: temp.java
229 Entering Extended Passive Mode (|||45598|)
150 Opening BINARY mode data connection for temp.java
100% |*****************************************************|  4526        2.28 MiB/s    00:00 ETA
226 Transfer complete
4526 bytes sent in 00:00 (1.23 MiB/s)
ftp> ls
229 Entering Extended Passive Mode (|||59249|)
150 Opening ASCII mode data connection for file list
-rw-r--r--   1 temporary temporary        0 Nov 29 01:24 amazingFile.bz2
-rwxrwxrwx   1 0        0            13204 Nov 28 20:03 download.jpg
-rw-r--r--   1 temporary temporary  8785767 Nov 28 23:21 EthicsFinalPresentation.pptx
-rwxrwxrwx   1 0        0                0 Nov 28 21:07 New.bz2
-rwxrwxrwx   1 0        0              525 Nov 28 20:03 project.html.bz2
-rw-r--r--   1 temporary temporary     1171 Nov 28 22:10 tbupldd.txt
-rw-r--r--   1 temporary temporary     4526 Dec  5 02:41 temp.java
-rwxrwxrwx   1 0        0              406 Nov 28 20:03 test.class
-rw-r--r--   1 temporary temporary     1171 Nov 28 21:05 ToDo.txt
226 Transfer complete
ftp>
```

Figure 5. The FTP server accessed through the CLI

Once the system was operational, we tested it by connecting to our FTP server via the Mac Command Line Interface (CLI) in the terminal program. As can be seen in Figure 5, the FTP server had full functionality, allowing all manner of FTP commands, though the main focus was on the *get* (to download files) and *put* (to upload files) commands. Once the FTP server's functionality was verified through the CLI, we connected to the server through a Web browser. As expected, the browser had to first log in with the required credentials before being able to access the files stored on the server. The login authentication may be seen in Figure 6, while the browser's user-interface for interacting with the file system may be seen in Figure 7.

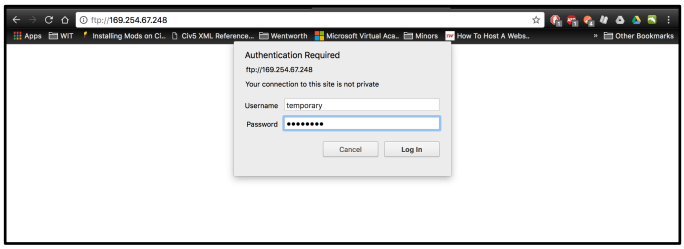

Figure 6. The FTP server login authentication

Index of /

Name	Size	Date Modified
amazingFile.bz2	0 B	11/28/17, 8:24:00 PM
download.jpg	12.9 kB	11/28/17, 3:03:00 PM
EthicsFinalPresentation.pptx	8.4 MB	11/28/17, 6:21:00 PM
New.bz2	0 B	11/28/17, 4:07:00 PM
project.html.bz2	525 B	11/28/17, 3:03:00 PM
tbupldd.txt	1.1 kB	11/28/17, 5:10:00 PM
test.class	406 B	11/28/17, 3:03:00 PM
ToDo.txt	1.1 kB	11/28/17, 4:05:00 PM

Figure 7. User-interface of browser

C. *HTTP Server*

With a web page ready for deployment as a user-friendly interface to the functional FTP server, and PHP code on deck to handle uploads to the FTP through the browser itself, we created our own HTTP server from scratch in Java using socket programming.

The Original design behind the Java code was to create a library with as much ease for the user as possible, while maintaining most, if not all, of the functionality. The user of the library needs only to implement a main() method with the same, or similar, functionality as seen in Figure 8.

```
HTTP testServer;
try {
    testServer = new HTTP("root/directory/ex_folder/folder_of_website");

    while (true){
        testServer.makeConnection();

        testServer.readMessage();

        testServer.interpretMessage();

        testServer.sendMessage();

        testServer.breakConnection();
    }

} catch (Exception e) {
    e.printStackTrace();
}
```

Figure 8. An example of the created HTTP library

HTTP server provides different services via different methods. For example, the interpretMessage() method interprets the message read from the client and forms a reasonably correct response message to return back to the client. The HTTP class uses many other helper classes and enums, such as the Message class (contains and maintains an HTTP message), to more easily structure and organize the server into pieces that don't often change.

Creating a fully-functioning HTTP server from scratch was not trivial. All HTTP interactions involve an HTTP message, including a start-line, optional header lines, and a data field (which could be empty). Much of the interaction between a Web browser and server occur through the optional header lines in each message, which allows the server and client to communicate how the other is operating as well as useful information about the data field [9].

With the HTTP server code written, tested, and debugged, we deployed both the HTTP and the FTP server applications onto the Raspberry Pi to host the web page and the files respectively, and, with the aid of the PHP code, wove them together to create a seamless user experience.

IV. Conclusion and Future Work

Knock-Off Drive was an attempt as a solution to the issue of oversaturation in cloud-based data storage. Many companies tend to offer "secure, fast, and responsive" cloud servers, but news of big companies being hacked and data being compromised has only increased over the past few years. *Knock-Off Drive* provides a solution to this issue by allowing a user to create their own customized cloud server in the pocket.

Our research approach starts from investigating the issues in cloud storage, surveying affordable hardware and software and implementing an engineering solution. Although the current implementation is not necessarily as secure as known-companies at this moment, it demonstrates the idea of providing encrypted files and sending messages over HTTPS protocol. For the future work, we are working on modularizing the system to make it easier to build and setup such that more and more users can easily build private and customized personal data storage.

References

[1] Internet World Stats – Internet Penetration Rates. https://www.internetworldstats.com/top25.htm

[2] Statista: Cloud storage usage of online adults in the U.S. https://www.statista.com/statistics/710964/us-cloud-computing-consumer-usage/

[3] 6 security risks of enterprises using cloud storage and file sharing apps. https://digitalguardian.com/blog/6-security-risks-enterprises-using-cloud-storage-and-file-sharing-apps

[4] Safety Concerns with Cloud Storage & Their Possible Solutions. https://smidcloud.com/en/safety-concerns-with-cloud-storage-their-possible-solutions/

[5] HTML Element Reference, W3schools, February 2017. https://www.w3schools.com/tags/default.asp

[6] CSS Reference, W3schools, February 2017. https://www.w3schools.com/cssref/default.asp

[7] JavaScript and HTML DOM Reference, W3schools, February 2017. https://www.w3schools.com/jsref/default.asp

[8] PHP 5 FTP Functions, W3schools, February 2017. https://www.w3schools.com/php/php_ref_ftp.asp

[9] How To Host A Website With Raspberry Pi, ReadWrite, June 2017. https://readwrite.com/2014/06/27/raspberry-pi-web-server-website-hosting/