2016

OSS Assessment Report

REEP.IO OSS ASSESSMENT

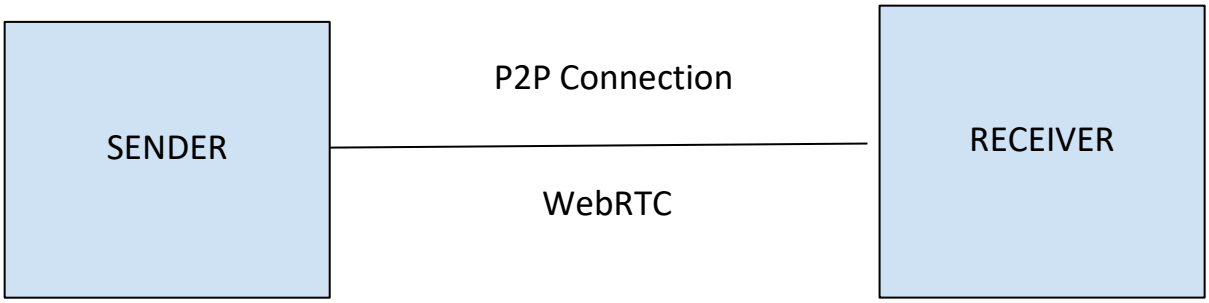
RAGHUL SOMINENI RAGHUPAHTY (RSR379) N11371498
ABHISHEK BALAJI VENKATARAAMAN SANGEETHA (AVS395) N17497529

Table of Contents

1. INTRODUCTION

https://reep.io is a browser based P2P file transfer platform. It allows secure transferring of files from the sender's system to the receiver's system. A peer to peer connection from the sender to the receiver system is created. Therefore, there is no concept of a server handling requests here and no metadata or contents of the file is stored in their server. It uses WebRTC to setup connection between the peers. (Fun Fact: the creators named the site reep by reversing peer.)

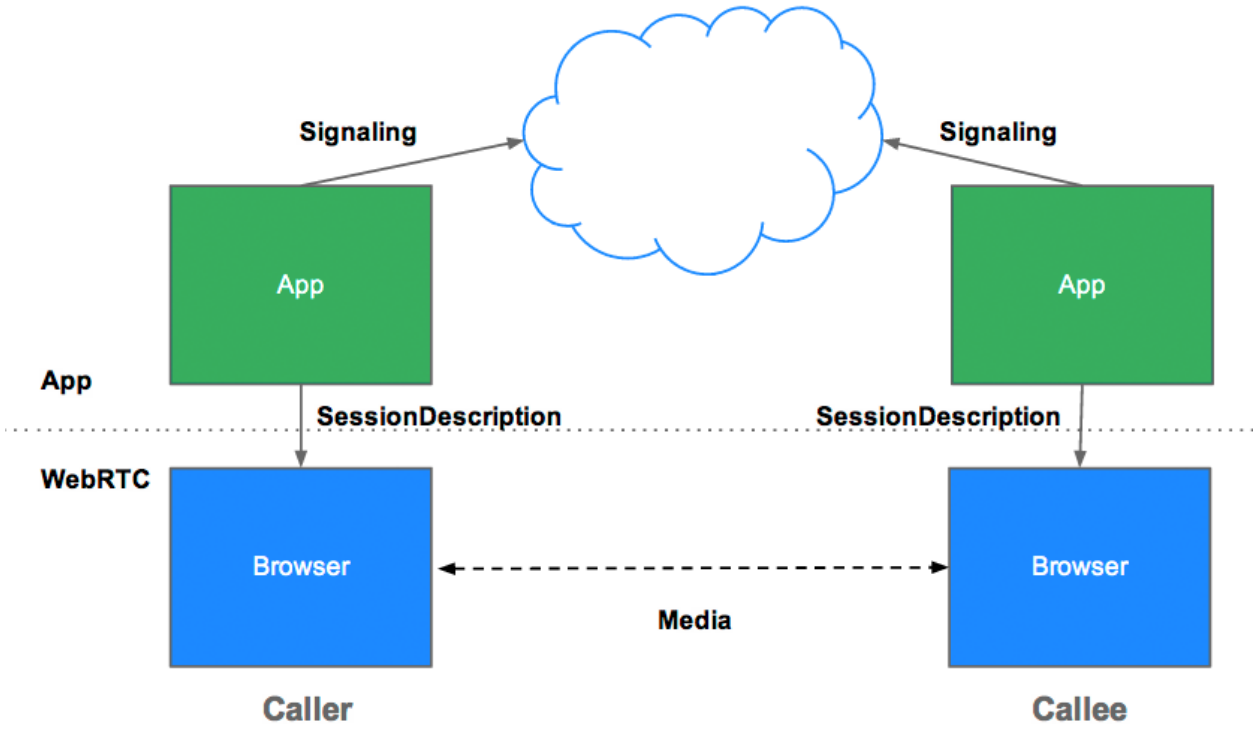
reep.io's only purpose is to make it easy for you to connect to peers by providing you with a link that automatically connects you both. After the initial handshake, we are out. The file transfer now only happens between you two without uploading and downloading the file to a server. The whole communication is encrypted using Datagram Transport Layer (DTLS) using SSL. This reduces the risk of man in the middle attacks.



The WebRTC APIs are designed to allow JS applications to create real-time connections with Audio, Video and other data channels directly between users via browsers, or to servers supporting the WebRTC protocols. It also leverages navigator.mozGetUserMedia() to get access to mic and camera data. The following section will have more details on the WebRTC protocol.

1.1. Peer-to-Peer Communications with WebRTC:

WebRTC or Web Real-Time Communication is a free, open source project that provides browsers and mobile applications with Real-Time communications (RTC) capabilities via simple APIs. It provides support for real time browser to browser communication applications such as voice calling, video calling and P2P file sharing. Chrome, Firefox and mobile browsers can interoperate with WebRTC while other browsers require downloading plugins for interoperation to work. Reep.io uses WebRTC to setup a peer to peer connection between the sender and receiver. The WebRTC communication is encrypted using the latest cryptographic algorithm present in our browsers. The aim of WEBRTC is: To enable rich, high quality RTC applications to be developed for the browser, mobile platforms and loT devices, and allow them to communicate via a common set of protocols.



WebRTC consists of several interrelated APIs and protocols which work together to support the exchange of data and media between two or more peers.

Before media/data can be exchanged, we need to link two peers together. This is accomplished using the RTCPeerConnection Interface. The RTCPeerConnection Interface represents a WebRTC connection between the local computer and a remote peer. It provides methods to connect to a remote peer, maintain and monitor the connection, and close the connection once it's no longer needed. Once the connection has been established, data/media transfer can be done between the peers using the RTCData Channel or the MediaStream interface.

WebRTC lets you use the connection between two peers to transmit arbitrary binary data back and forth. This is accompanied using the RTCDataChannel interface. This can be used for back-channel information, or even as y our primary channel for exchanging any kind of data you wish. For games, it could be used to implement multiplayer support, transmitting player movement updates and the like back and forth. The MediaStream Interface represents a stream of media data being transmitted from one peer to another. This stream consists of one or more tracks; typically, this is an audio track and a video track. A stream can transmit live media (for audio calls or video conferencing) or stored media (such as a streamed movie). Now let's talk about the signaling channel that is needed to establish a connection. Unfortunately, WebRTC can't create connections without some sort of server in the middle. We call this the signal channel. It's any sort of channel of communication to exchange information before setting up a connection, whether by email, post card or a carrier pigeon... it's up to you.

The information we need to exchange is the offer and the Answer which just contains the SDP mentioned below.

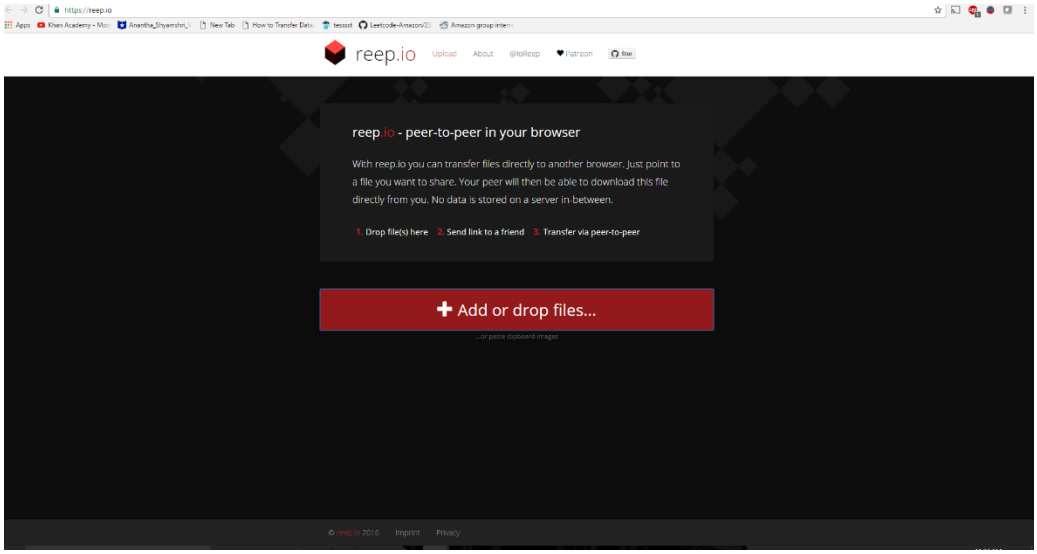
Peer A who will be the initiator of the connection, will create an offer. They will send this offer to Peer B using the chosen signal channel. Peer B will receive the offer from the signal channel and create an Answer. They will then send this back to Peer A along the signal channel. The configuration of an end point on a WebRTC connection is called a **session description.** The description includes information about the kind of media being sent, its format, the transfer protocol being used, the end point's IP address and port, and other information needed to describe a media transfer end point. This information is exchanged and stored using the **Session Description Protocol (SDP).**

2. UX, Design, and Ease of Use:

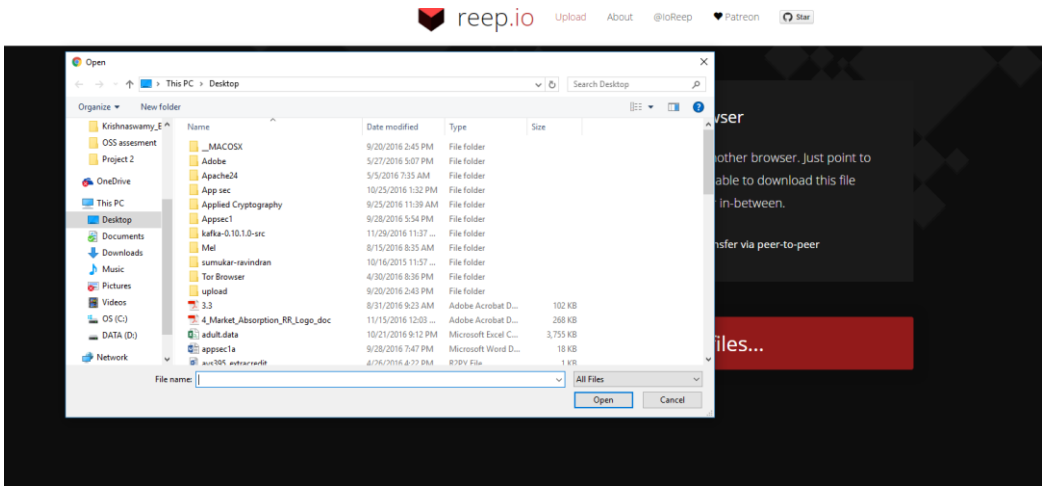
The user experience is very simple and minimal. The user can either select files by dragging them on to the website's page, or press the "Add or drop files.." button and open a file browser. If the user decides to upload more files, then they will all be shown on the same screen ordered vertically. Do note that it is key that the browser (or tab) is not closed if the file upload is in progress, else the file transfer will fail. The website is easy to use, and the process of uploading the file is the same as it has been so there is nothing new here. The user knows exactly what to do. The instructions are given clearly in the page of the website.

2.1. Using reep.io:

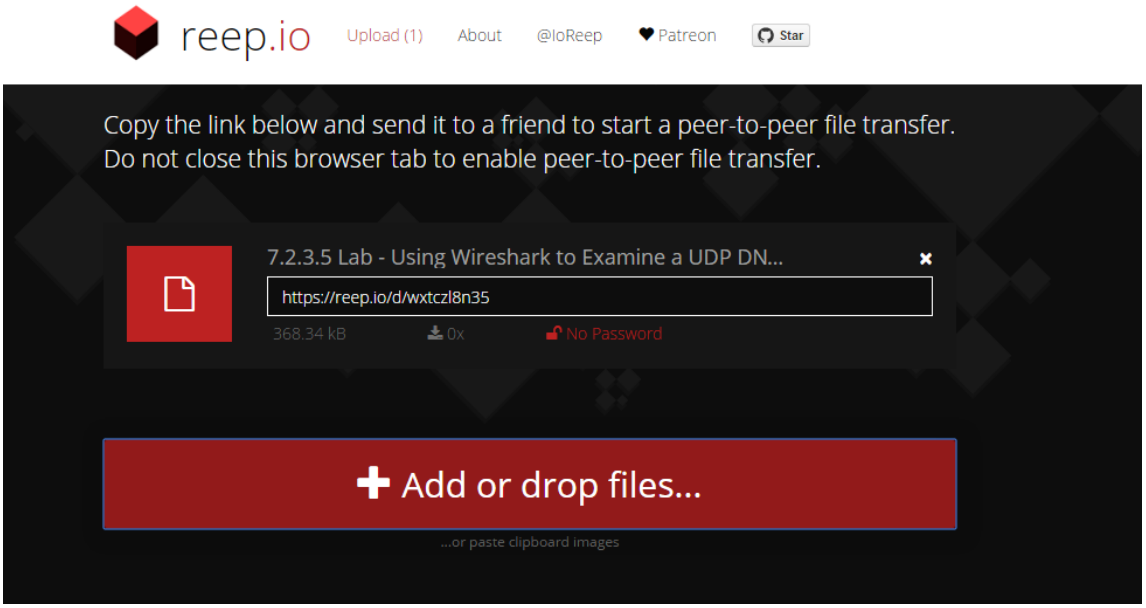
1) Open https://reep.io/



2) Choose the file you wish to upload



3) A link is created, which when sent to the receiver can be used to establish peer to peer connection for file transfer

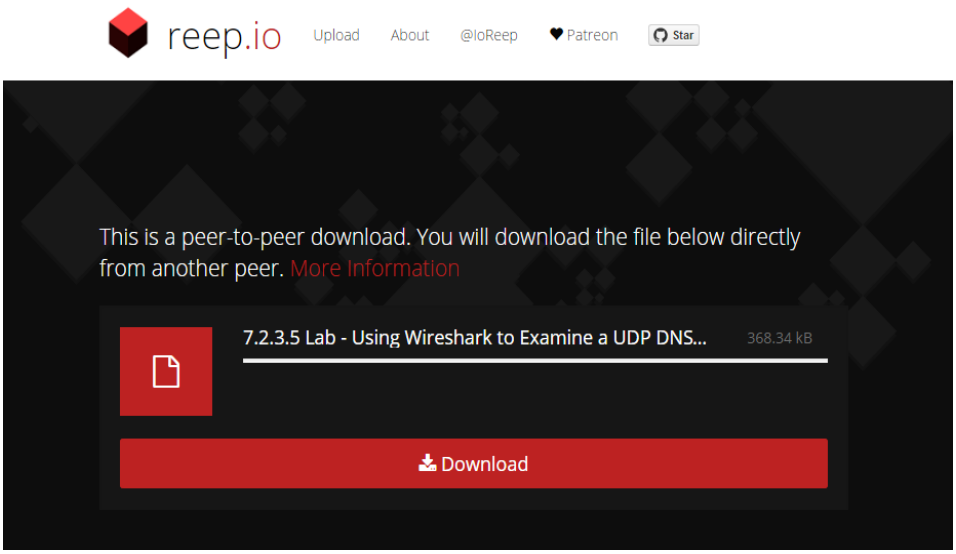


Note: Browser tab must not be closed

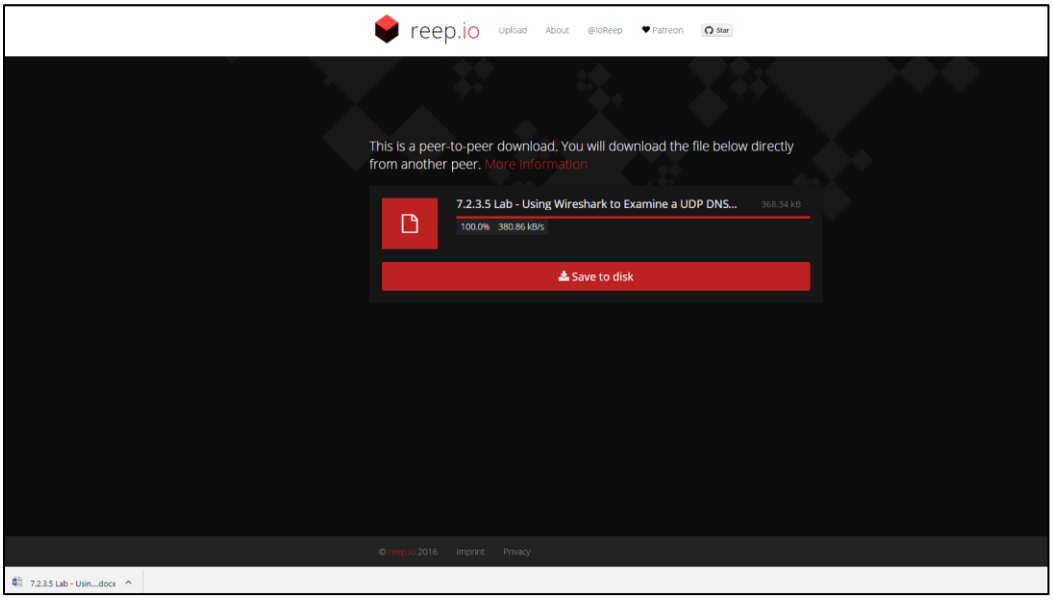
You can set a password if you want using the option provided before sharing the link.

The link is created by using two secrets one from the client (sender) and the other from the reep.io server.

4) When that link is opened, we get prompted with the download button



5) After the file is downloaded



3. Understanding reep.io :

What are the types of files reep.io can transfer?

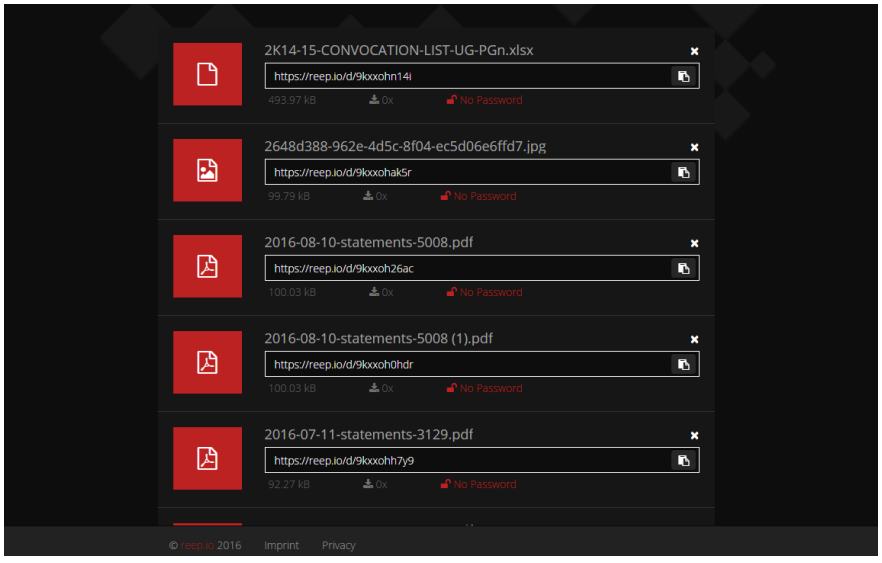
We tried sending different types of files ranging from .docx, .txt, .rar and image formats and reep.io worked without a hitch.

What is the number of files that can be transferred in one connection?

Though there is no strict file size limit that has been specified, the maximum number of files that can be uploaded at once stands at 8 which can be seen from reep.io's source code.

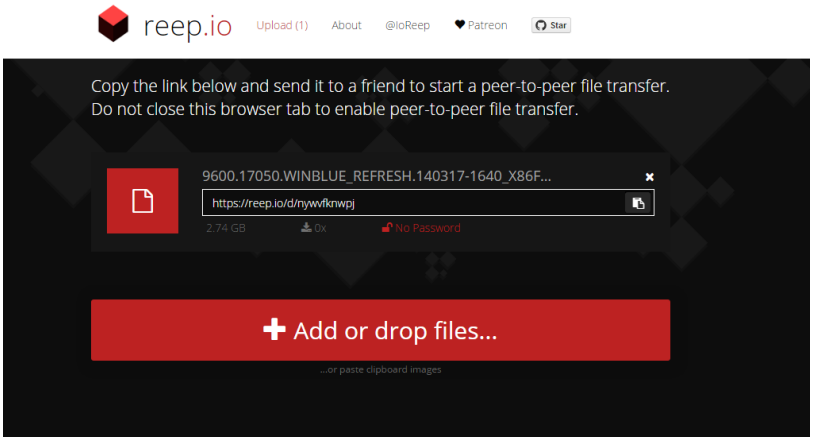
```
21          </section>
22
23          <div id="drop-box" class="ng-hide" data-p2m-drop-zone data-max-files="8" data-files="fileModel.files" data-ng-animate data-ng-class
24              <div>
25                  <span class="glyphicon glyphicon-plus"></span>
26                  Drop files here...
27              </div>
```

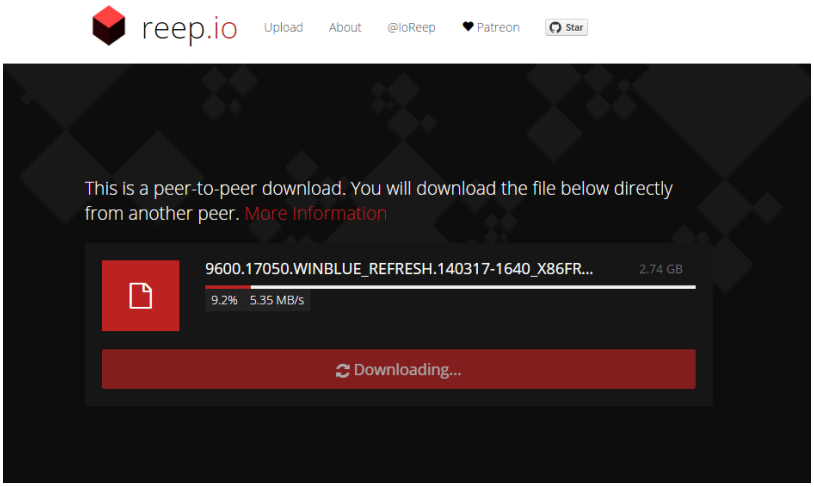
It would be futile to try crash the site with huge files because the server is not in play after initial handshake is finished. Reep.io generates new links for each file we try to upload.



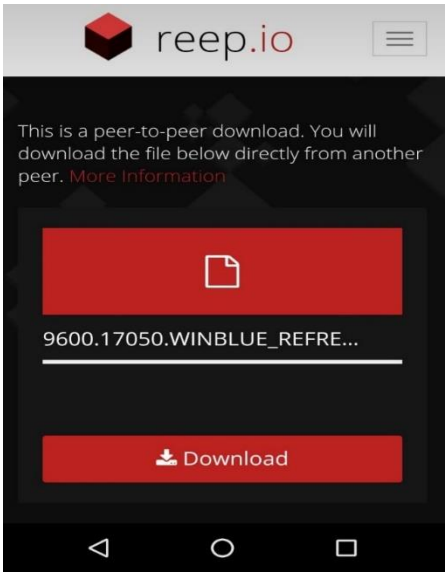
This makes it difficult to bombard the receiver with multiple files at the same time.

We tried transferring a 2.5gb file to more than one client (different tabs and on mobile)



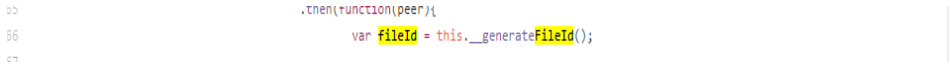


Tried it on a mobile web browser too:

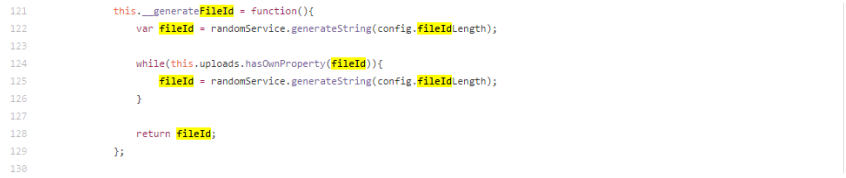


3. 1. How the URL is created:

First a random file id is generated which we can see from the below screenshots



```
.then(function(peer){
    var fileId = this.__generateFileId();
```

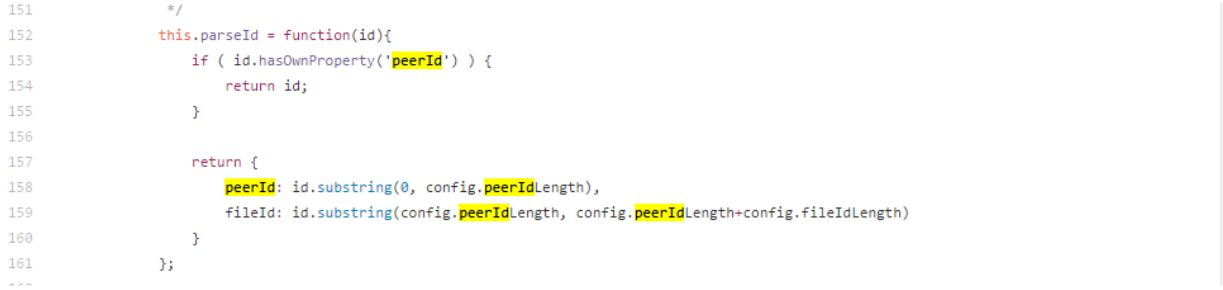


```
this.__generateFileId = function(){
    var fileId = randomService.generateString(config.fileIdLength);

    while(this.uploads.hasOwnProperty(fileId)){
        fileId = randomService.generateString(config.fileIdLength);
    }

    return fileId;
};
```

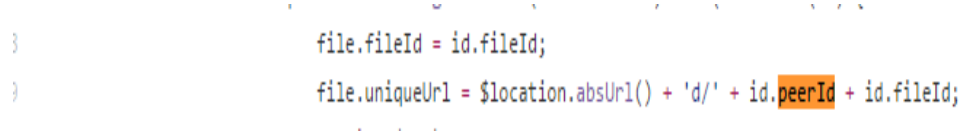
And a peerID is generated for each client



```
 */
this.parseId = function(id){
    if ( id.hasOwnProperty('peerId') ) {
        return id;
    }

    return {
        peerId: id.substring(0, config.peerIdLength),
        fileId: id.substring(config.peerIdLength, config.peerIdLength+config.fileIdLength)
    }
};
```

The unique URL is then obtained by combining the peer id and file id



```
file.fileId = id.fileId;
file.uniqueUrl = $location.absUrl() + 'd/' + id.peerId + id.fileId;
```

3. 2. Other Information:

1) If the Sender's browser is closed during transfer then the URL is deleted and the receiver can't download the file with the same URL as before, the sender must upload file again and share a new URL. The receiver will try to establish connection until refreshed.

2) If the receiver's browser is closed while transferring, the status is saved and download can resume with the same URL given that the sender hasn't closed the connection

3) One link can be shared with more than one receivers, if the sender's browser is not closed.

We tried downloading from more than two receivers at the same time and reep.io worked without a problem.

4. Vulnerabilities:

4. 1. WebRTC:

WebRTC is a new and trending technology for real time communication using browsers. However, the open-source nature of the technology may have the potential to cause security-related concerns to potential adopters of the technology. WebRTC is being developed every day and it has had just 2 major security threats as per CVE website.

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2016	2	2			1										
Total	2	2			1										
% Of All		100.0	0.0	0.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

Description of the DoS attack CVE-2016-1976

Vulnerability Details : CVE-2016-1976

Use-after-free vulnerability in the DesktopDisplayDevice class in the WebRTC implementation in Mozilla Firefox before 45.0 on Windows might allow remote attackers to cause a denial of service or possibly have unspecified other impact via unknown vectors.

Publish Date : 2016-03-13 Last Update Date : 2016-12-02

It also specifies what products are affected and these include Mozzila and WebRTC

Description of DoS attack with memory corruption CVE-2016-1975:

Reading more about WebRTC let me to know that **real IP addresses of the users** can be known even if they use a VPN. The security glitch affects WebRTC-supporting browsers such as Google Chrome and Mozilla Firefox, and appears to be limited to Windows operating system only, although users of Linux and Mac OS X are not affected by this vulnerability.

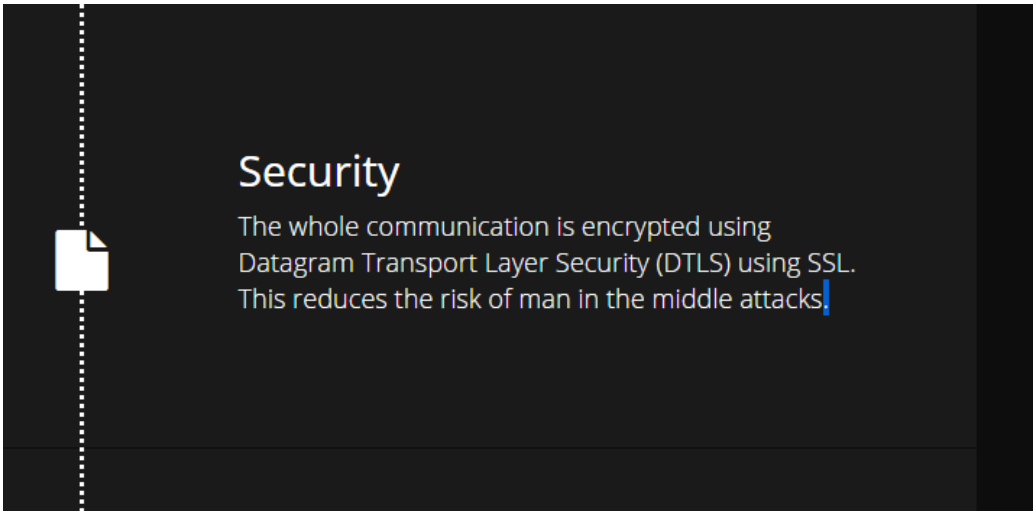
How does this work?

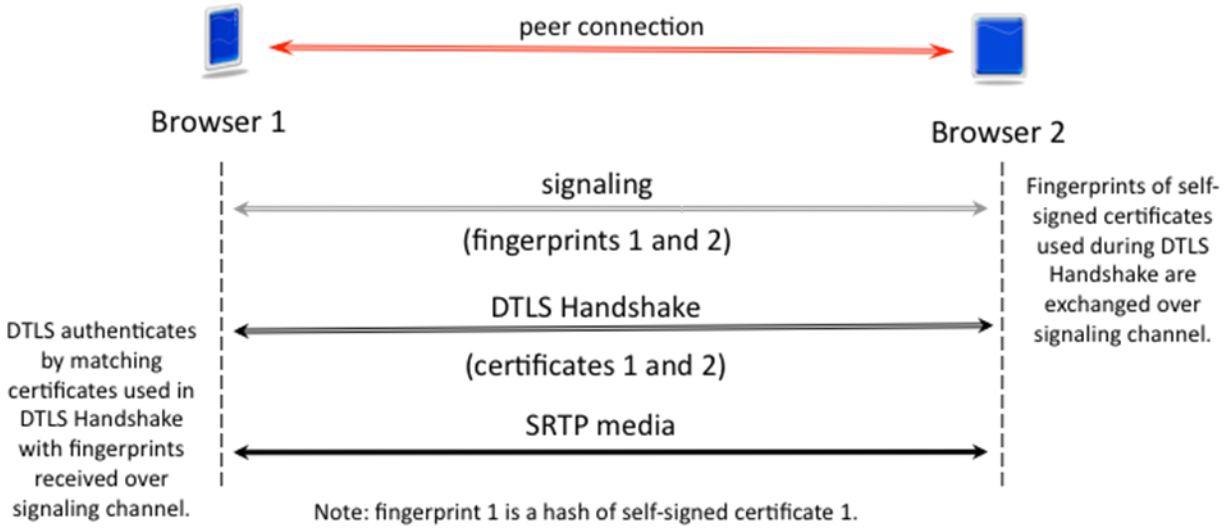
WebRTC allows requests to be made to STUN (Session Traversal Utilities for NAT) servers which return the "hidden" home IP-address as well as local network addresses for the system that is being used by the user.

Only consolation is, they can be accessed only using webRTC and java script and not through developer console because the requests are made outside normal XML/HTTP request procedure.

4.2. Security Vulnerabilities:

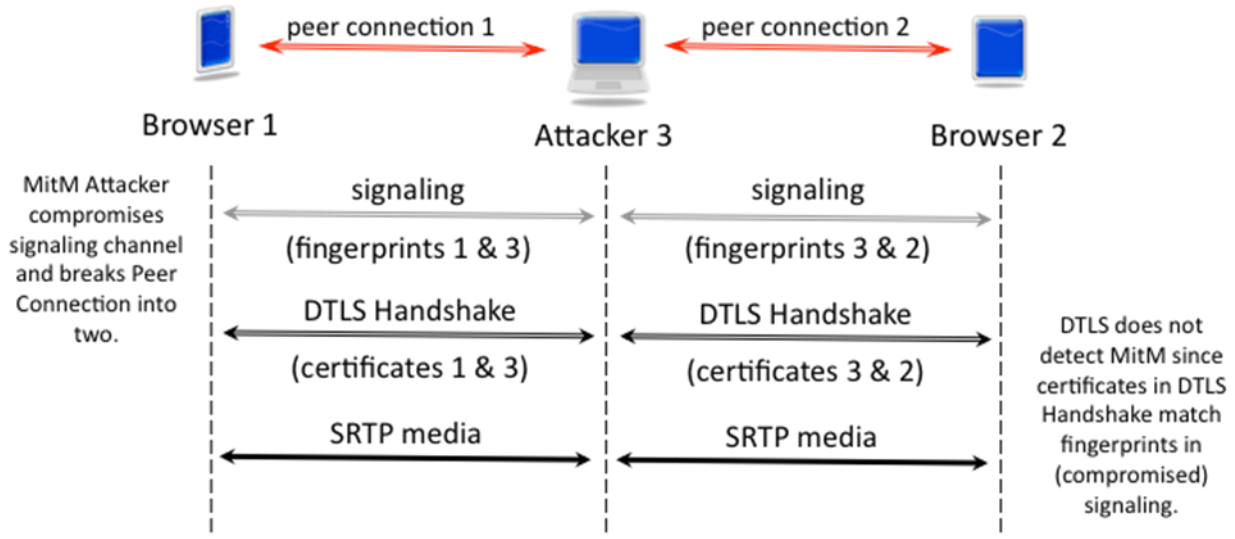
Reep.io uses DTLS with SSL to encrypt the communication, this is done to reduce man in the middle attacks.





Fingerprint here refers to the sharing of public keys between the browsers

Though DTLS can be used to “reduce” MITM attacks, it is not however fully secure from it.



The attacker can use his fingerprint instead of browser1 and establish a connection, DTLS will not detect any anomaly because the attacker’s connection is legit as far as browser 2 is concerned.

The person who came up with this attack is Alan Johnston gave two reasoning as to why the attack takes place.

1) It was easy to develop new ideas in webRTC and to place the node.js file on the server side.

2) WebRTC uses DTLS to produce keys for the SRTP media session but the normal protections offered by TLS are not present.

4.3. File Security:

To check If the file has been modified or to check if the link is authentic, reep.io uses CRC tables to check for changes in the raw data during transmission. But this is not that efficient because there are many vulnerabilities which can be used against the website. One of such is the compensation attack detection vulnerability.

A design flaw in the SSH1 protocol (protocol 1.5) that could lead an attacker to inject malicious packets into an SSH encrypted stream that would allow execution of arbitrary commands on either client or server. The problem was not fixable without breaking the protocol 1.5 semantics and thus a patch was devised that would detect an attack that exploited the vulnerability found. The attack detection is done in the file deattack.c from the SSH1 source distribution. A vulnerability was found in the attack detection code that could lead to the execution of arbitrary code in SSH servers and clients that incorporated the patch.

There is a JavaScript file working which prevents file to be downloaded by crawler programs, this is a feature that needs to be noted.

4.4. Application can be misused to send Malicious files:

It was easy to send a malicious file (obtained from the internet) to another browser, as no file checks are done to see If a file is malicious. This could be misused to a great extent. An attacker can masquerade as a legitimate sender and send the client the malicious file which could do anything the attacker wants it to do. The attacker can affect multiple systems with this and from there he can use the same method to spread it to even more clients.

4.5. How can the URL be sent to the receiver?

The URL must be sent to the receiver by the sender for the receiver to use that URL to download the file. The receiver can use various ways to do that like email, instant messaging services etc. This further widens the attack surface as the attacker just should get this link to download the file. Every option has its own risk attached to it. For example, if the sender decides to use SMS to send the url, an attacker can be eavesdropping the communication and thereby has the link through which he can download it, furthermore, the attacker can change the link being sent to the receiver thereby performing MITM and dos attacks. Thus, sharing of the link must be done carefully.

5. Ideas for improvement:

- The file transfer did not work for us on Mozilla Firefox, but it worked on google chrome. The developer should make this app cross platform, or it will not be of much use to the others.
- Reep io is session based, so once the session is disconnected the transaction is lost, the file transfer must be done from scratch. This is a pain if you are uploading a large file. Other file transfer techniques such as FTP, HTTP file upload, skype file upload are all session based, but if the developer finds a way to cache the data download and keeps track of what part of the data has been uploaded and what is remaining, we are sure that his will surely make the app more appealing and more useful to others.
- We also noticed that if we upload a video file, the file type (in the name) is not changed back to the original type. Maybe this was an edge case, but the developer should consider this as an area of development.
- Each file generates a new URL that needs to be shared, though it could be viewed as something which decrease threats, it becomes a pain to send each link separately. An option to group files that needs to be sent would be a great improvement.
- The files are just sent as raw data as such, this could be vulnerable to an attacker who is listening to either one of the client, hence a new procedure to protect the files must be set, our suggestion is to use hashes to protect the file further.
- A safe way to send the url if provided by the site itself would improve the security of the application a great deal.
- If the URLs not valid, no error message is displayed, the url just contains “null”, displaying that the file hasn’t been used to generate correct URL would be useful for end clients.
- If the URL has been deleted the client just retries to download the file over and over until timeout is reached, Checking the URL to see if it is valid would reduce this.

6. Conclusion:

The idea of the app is good, but not novel. There are many peer to peer file transfer websites available out there, and this feature is already integrated in other software's such as skype. We recommend that if the developers work on the above-mentioned problems, they can rise above the present competition. Just because the base idea of the app is peer to peer transfer, it doesn't mean that caching of the file cannot be done. Caching is the one feature which we feel that must be done and one which is very much needed for the app's success. We really feel that the clients using the app need not start their file transfer all over again if their session has been disconnected. This is one problem that is there many other file transfer services and the one on which the developers should be working. We believe that this is the reason torrents and some file download managers are so popular that they cache the data, and continue the download from the place where they were interrupted.

7. References:

1. http://thehackernews.com/2015/02/webrtc-leaks-vpn-ip-address.html
2. http://blog.webrtc-solutions.com/webrtc-security-concerns/
3. https://en.wikipedia.org/wiki/Cyclic_redundancy_check
4. https://www.coresecurity.com/content/ssh1-crc-32-compensation-attack-detector-vulnerability