OSS ASSESMENT – Bitfly

Preethi Raghunathan

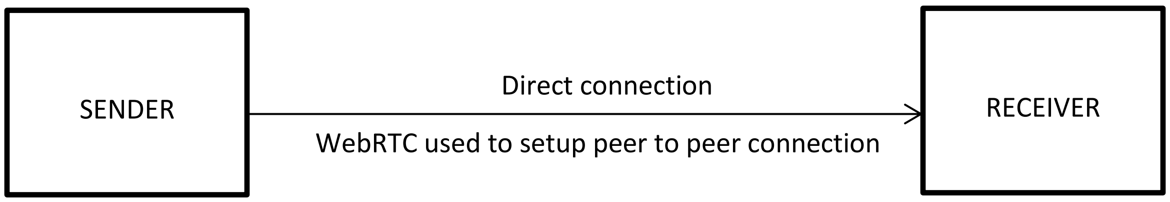
https://github.com/bitfly-p2p

CONTENTS

Bitfly OSS ASSESMENT

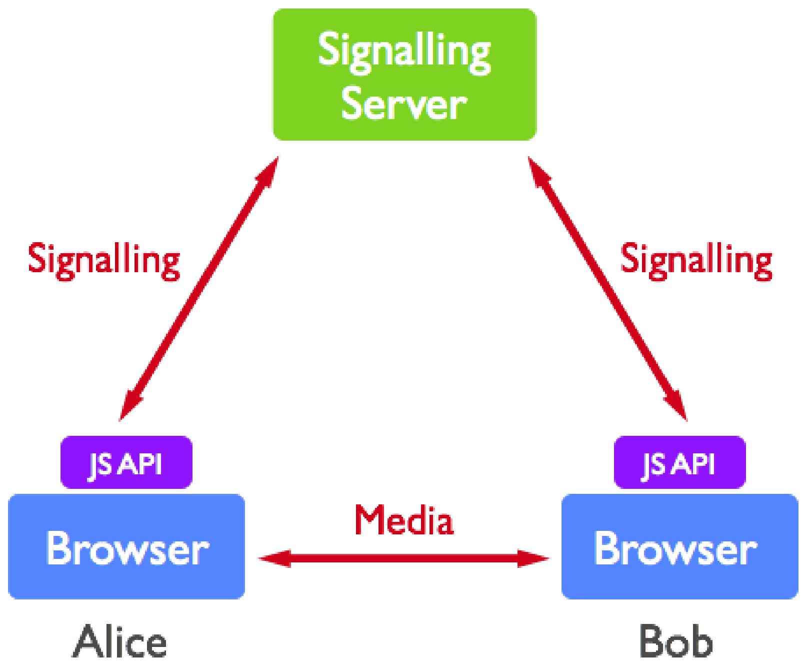
INTRODUCTION:

Bitfly is an application that allows for a secure way of transferring files from the sender's system to the receiver's system. It creates a peer to peer connection from the sender to the receiver system. Therefore, there is no concept of a server handling requests here. According to what they say, no metadata or contents of the file is stored in their server.



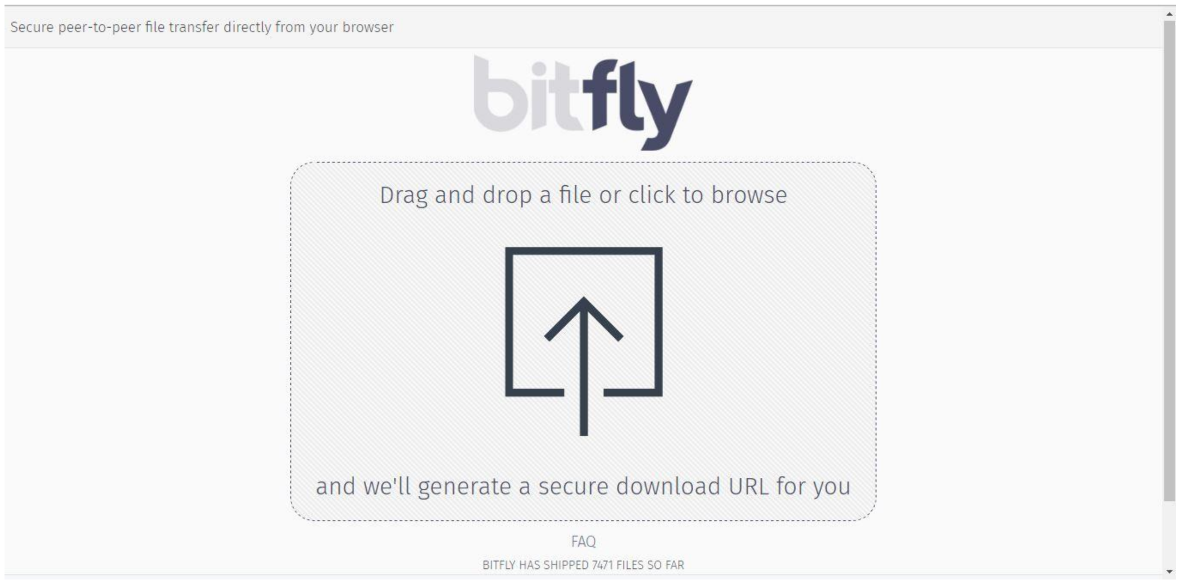
WEBRTC: [6]

WebRTC or Web Real-Time Communication is an open source project that provides support for real time browser to browser communication applications such as voice calling, video calling and P2P file sharing. Chrome and Mozilla can interoperate with WebRTC while other browsers require downloading plugins in order for interoperation to work. Bitfly uses WebRTC in order 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.

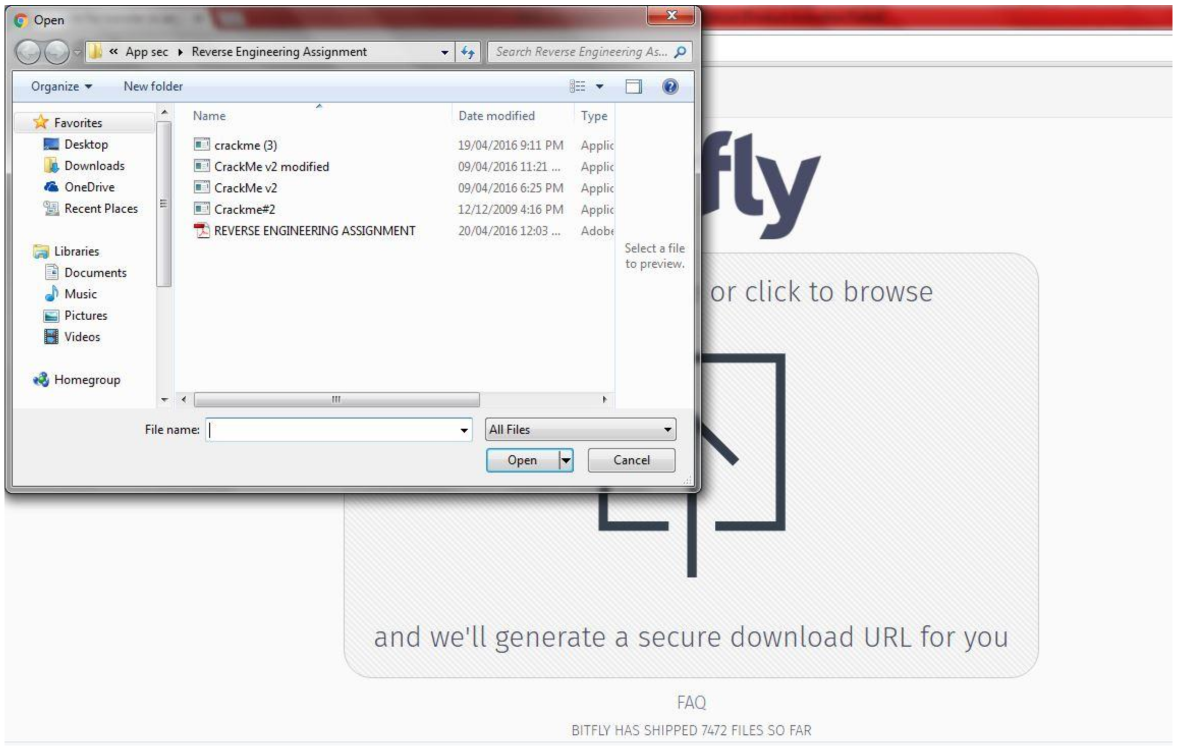


WORKING OF THE APPLICATION:

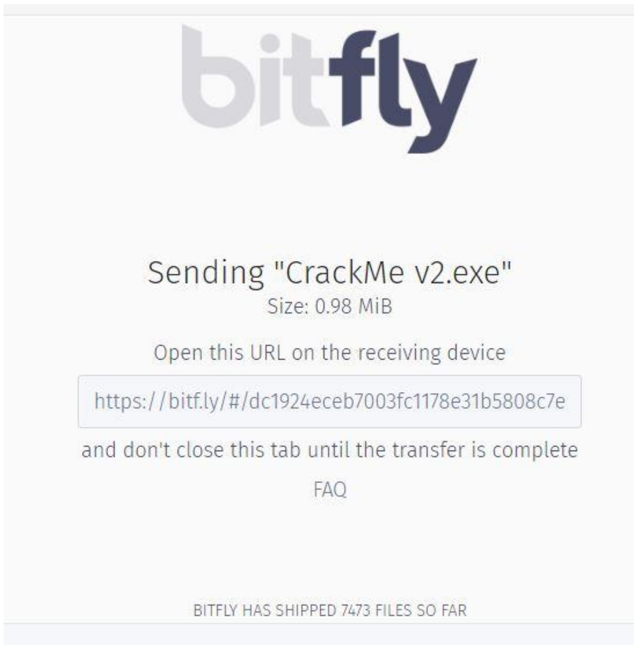
1. We go to the site (https://bitf.ly).


Secure peer-to-peer file transfer directly from your browser
bitfly
Drag and drop a file or click to browse
and we'll generate a secure download URL for you
FAQ
BITFLY HAS SHIPPED 7471 FILES SO FAR

2. We upload the file to be transmitted using the link provided.

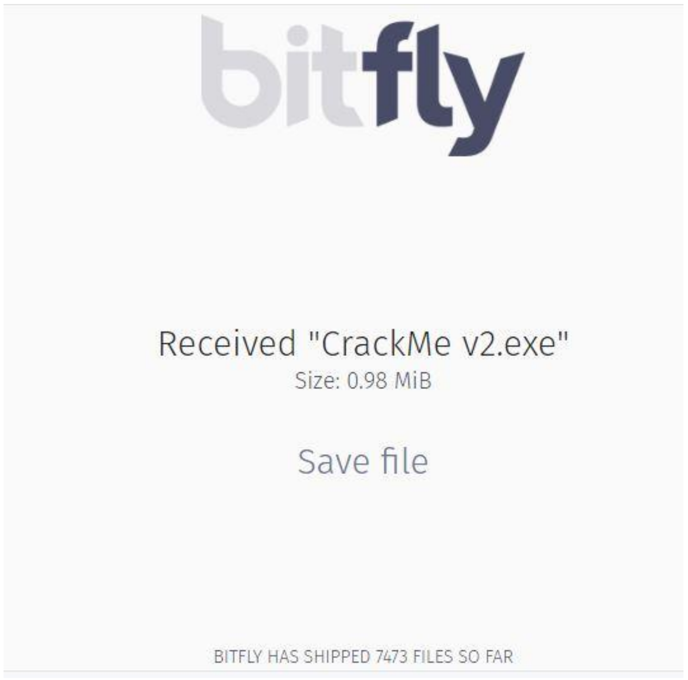


3. A secret URL link is created through which the receiver can obtain the file. This URL must be sent to the receiver by the sender through some means such as email.

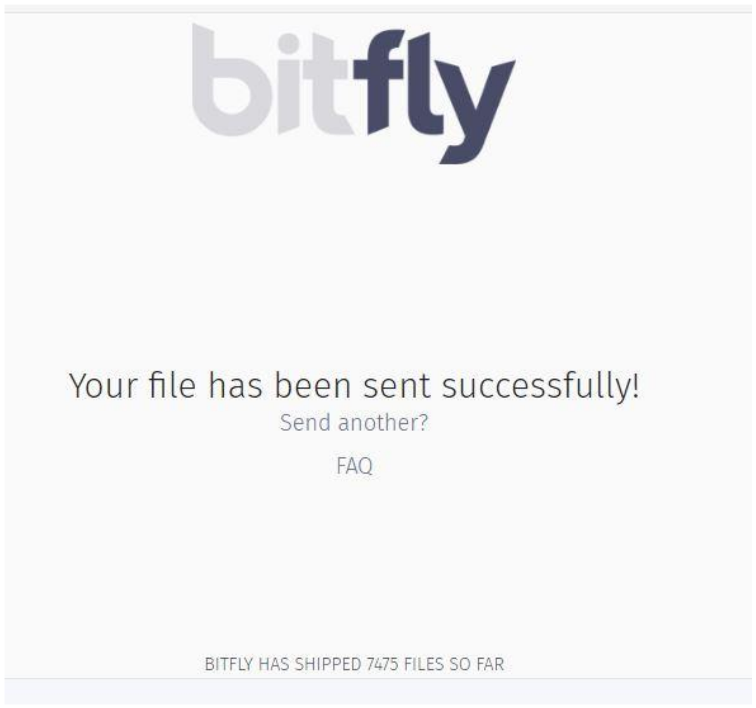

Sending "CrackMe v2.exe"
Size: 0.98 MiB
Open this URL on the receiving device
https://bitf.ly/#/dc1924eceb7003fc1178e31b5808c7e
and don't close this tab until the transfer is complete
FAQ
BITFLY HAS SHIPPED 7473 FILES SO FAR

The URL contains a secret 32 byte string which is used to access the file. **This 32 byte string is formed from two secrets, one created at the client and the other created at the Bitfly server.**

4. Opening that secret link provides us with the download option for the file.


Received "CrackMe v2.exe"
Size: 0.98 MiB
Save file
BITFLY HAS SHIPPED 7473 FILES SO FAR

5. When the file has successfully reached the receiver a success message is displayed.



Your file has been sent successfully!

BITFLY HAS SHIPPED 7475 FILES SO FAR

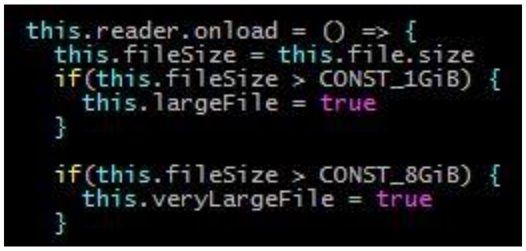
UNDERSTANDING THE APPLICATION:

Can it accept any file format for transmission?

I tried sending different types of files to see if all are accepted or if there are problems with some file types. The file types used were .docx, .rar, .xls, .mp4, .py and .jpg. All of these were accepted and the process of sending the files was carried out without any problems.

Maximum size of a file that can be transferred (Check for possible Denial Of Service attack due to size of file):

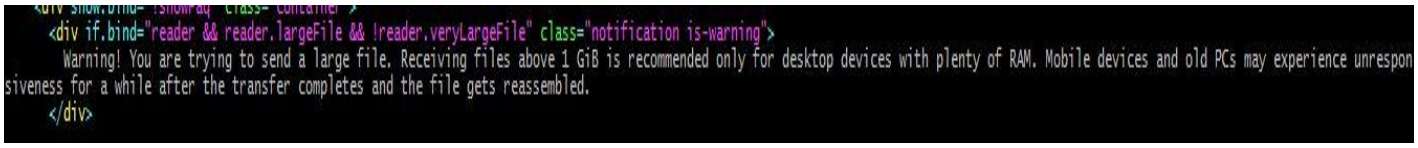
There is no limit specified for the size of the file that is to be transmitted. Therefore, it can transfer files of any size. The larger the file the more time it takes to be transmitted. There are checks put in place warning the user what will happen if the user decides to send files of larger sizes.



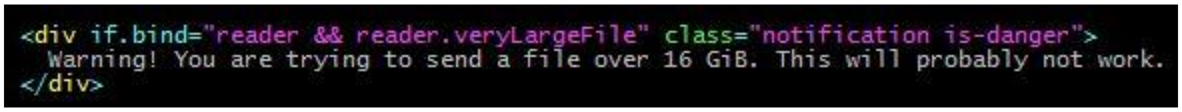
```
this.reader.onload = () => {
  this.fileSize = this.file.size
  if(this.fileSize > CONST_1GiB) {
    this.largeFile = true
  }

  if(this.fileSize > CONST_8GiB) {
    this.veryLargeFile = true
  }
```

From the above image we can infer that it considers files greater than 1 GB to be large files and files greater than 8GB to be very large files.

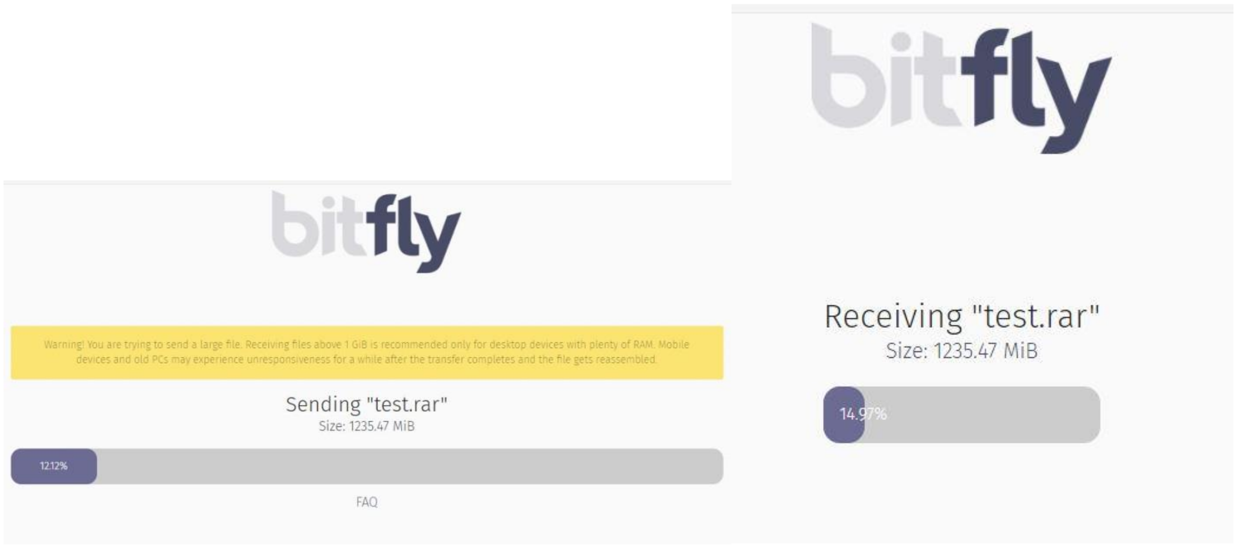


```
    <div if.bind="reader && reader.largeFile && !reader.veryLargeFile" class="notification is-warning">
      Warning! You are trying to send a large file. Receiving files above 1 GiB is recommended only for desktop devices with plenty of RAM. Mobile devices and old PCs may experience unrespon
siveness for a while after the transfer completes and the file gets reassembled.
    </div>
```



```
<div if.bind="reader && reader.veryLargeFile" class="notification is-danger">
  Warning! You are trying to send a file over 16 GiB. This will probably not work.
</div>
```

I was curious to know if I could bring down this site by sending multiple large size files at the same time. We can send only one file at a time. This application doesn't provide us with an option to send multiple files. The only way to send multiple files to perform this test is to open the website in multiple tabs or windows, the number of them being the number of files to be sent. This works because when we visit the site in another tab or window while a file is being sent we get the main page which allows us to send other files. I tried sending a .rar file that is 1.20 GB in size to 10 recipients (my own computer) simultaneously.



Warning! You are trying to send a large file. Receiving files above 1 GiB is recommended only for desktop devices with plenty of RAM. Mobile devices and old PCs may experience unresponsiveness for a while after the transfer completes and the file gets reassembled.

Sending "test.rar"
Size: 1235.47 MiB
12.12%
FAQ

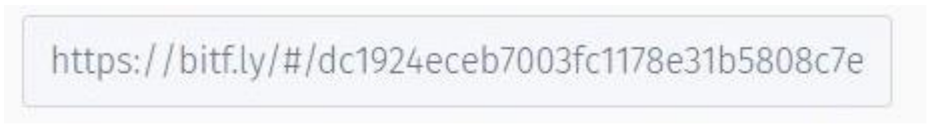
Receiving "test.rar"
Size: 1235.47 MiB
14.97%

Reason why DOS is not possible:

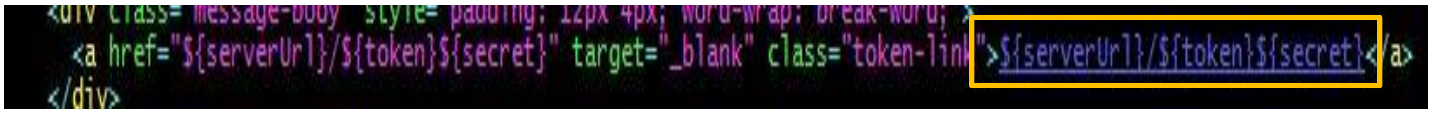
The application uses a peer to peer connection instead of client/server architecture to transmit files. Therefore, there is nothing like overloading the server with requests or large data to bring down the site.

Secret URL analysis:

As mentioned before, the secret URL generated to access the file consists of secrets generated by both the server and the client.



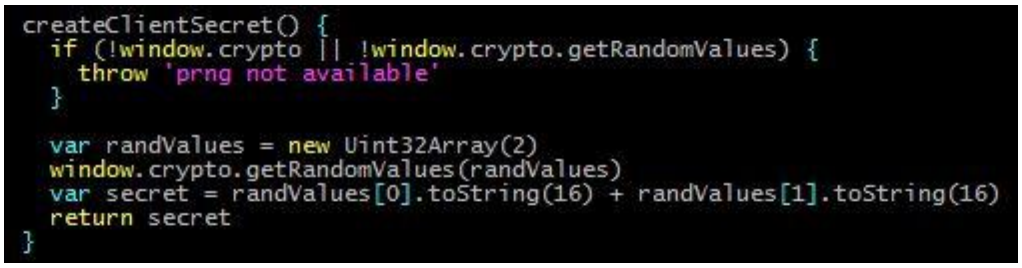
```
https://bitf.ly/#/dc1924eceb7003fc1178e31b5808c7e
```



```
<div class="message-body" style="padding: 12px 4px; word-wrap: break-word;">
  <a href="${serverUrl}/${token}${secret}" target="_blank" class="token-link">${serverUrl}/${token}${secret}</a>
</div>
```

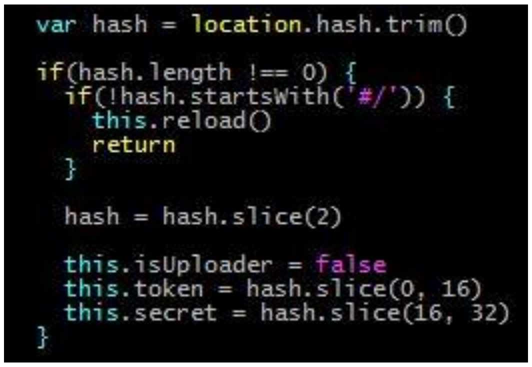
According to the above given code the URL is crafted for the file as follows.

<serverURL>/<token><secret>



```
createClientSecret() {
  if (!window.crypto || !window.crypto.getRandomValues) {
    throw 'prng not available'
  }

  var randValues = new Uint32Array(2)
  window.crypto.getRandomValues(randValues)
  var secret = randValues[0].toString(16) + randValues[1].toString(16)
  return secret
}
```

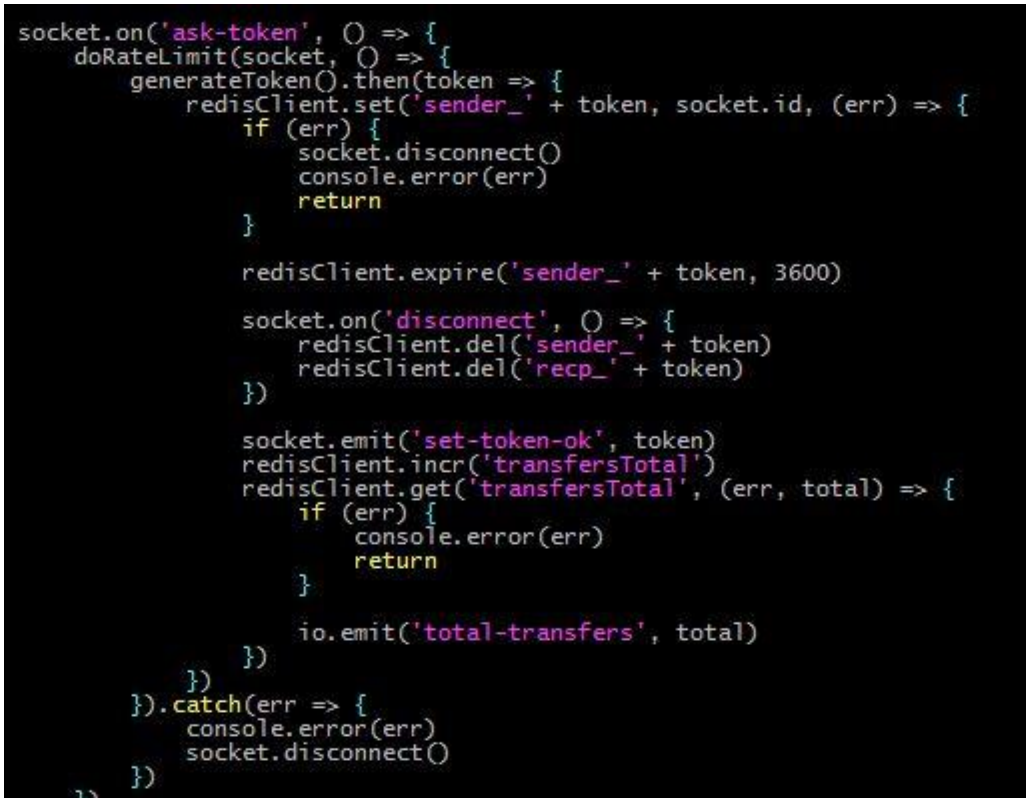


```
var hash = location.hash.trim()

if(hash.length !== 0) {
  if(!hash.startsWith('#/')) {
    this.reload()
    return
  }

  hash = hash.slice(2)

  this.isUploader = false
  this.token = hash.slice(0, 16)
  this.secret = hash.slice(16, 32)
}
```



```
socket.on('ask-token', () => {
    doRateLimit(socket, () => {
        generateToken().then(token => {
            redisClient.set('sender_' + token, socket.id, (err) => {
                if (err) {
                    socket.disconnect()
                    console.error(err)
                    return
                }

                redisClient.expire('sender_' + token, 3600)

                socket.on('disconnect', () => {
                    redisClient.del('sender_' + token)
                    redisClient.del('recp_' + token)
                })

                socket.emit('set-token-ok', token)
                redisClient.incr('transfersTotal')
                redisClient.get('transfersTotal', (err, total) => {
                    if (err) {
                        console.error(err)
                        return
                    }

                    io.emit('total-transfers', total)
                })
            })
        }).catch(err => {
            console.error(err)
            socket.disconnect()
        })
```

Using secrets from both ends is a good idea because this provides more security. An attacker will have to get both in order to get access to the file.

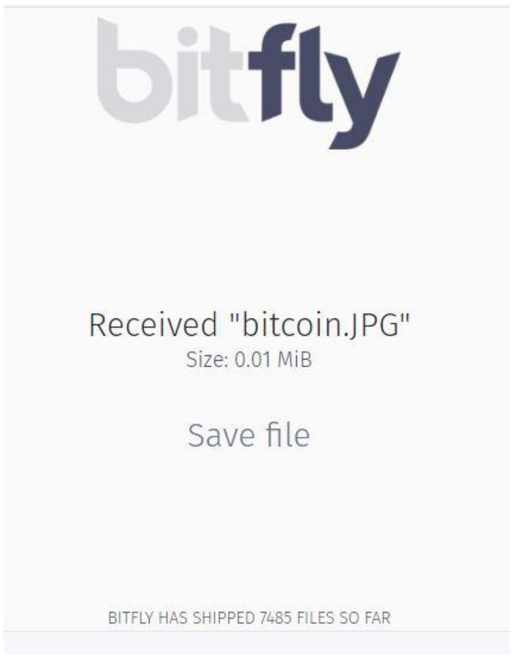
File protections set in place:

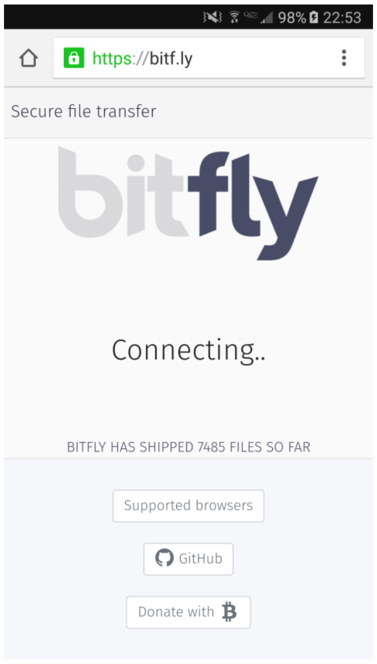
- If the browser window on the sender's side is closed then the generated URL for that file becomes invalid. When the user opens the link the user will be redirected to the main page of the website where we can upload files and get a URL for the receiver to access it.
- If the link is simultaneously being opened on two different machines then one of two things can happen.

    1. One of the users who may have been the first to request the file beating the other user by milliseconds could gain access to the file.
    2. Both users will not get access to the file and they will keep waiting for a connection.

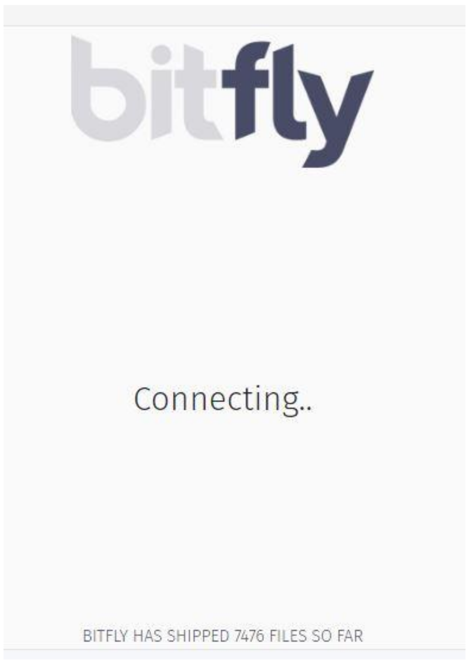
Test: I obtained a URL for a test file and decided to open the file in my laptop and in my mobile phone.

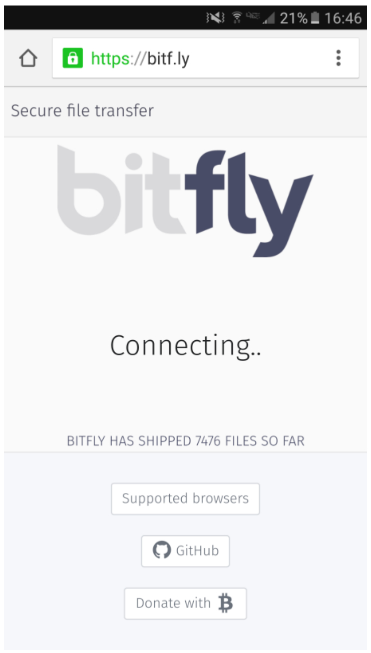
Case 1: Trying to open the link in both devices with a gap of less than a second.





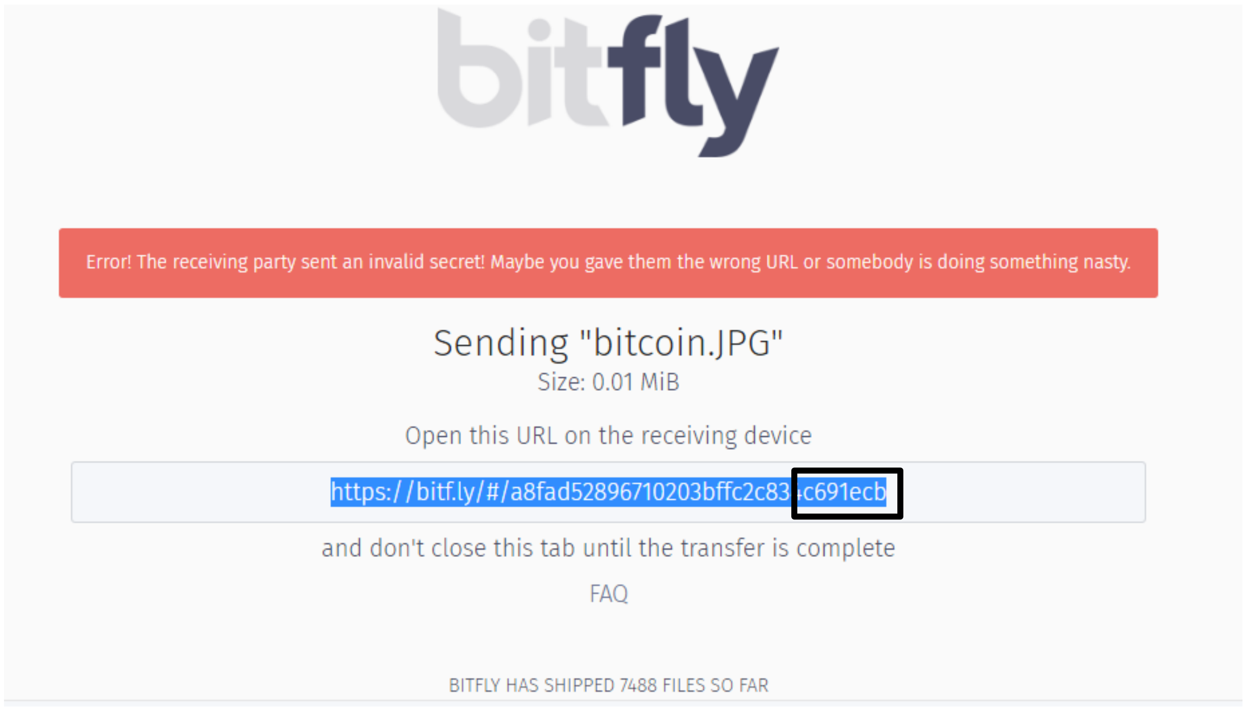
Case 2: I had a friend click on the link in my laptop while I opened it in my phone for synchronization purposes.





- Once the URL to the file is opened it becomes invalid and cannot be used to access the file later. So the URL serves as a one-time use only. If used again we get the window shown in the previous point which appears to be trying for a connection but does not go through.

- If the secret key in the URL is different, the following error message is sent to the sender and at the receiver's end the website keeps trying for a connection.



Error! The receiving party sent an invalid secret! Maybe you gave them the wrong URL or somebody is doing something nasty.

Sending "bitcoin.JPG"

Size: 0.01 MiB

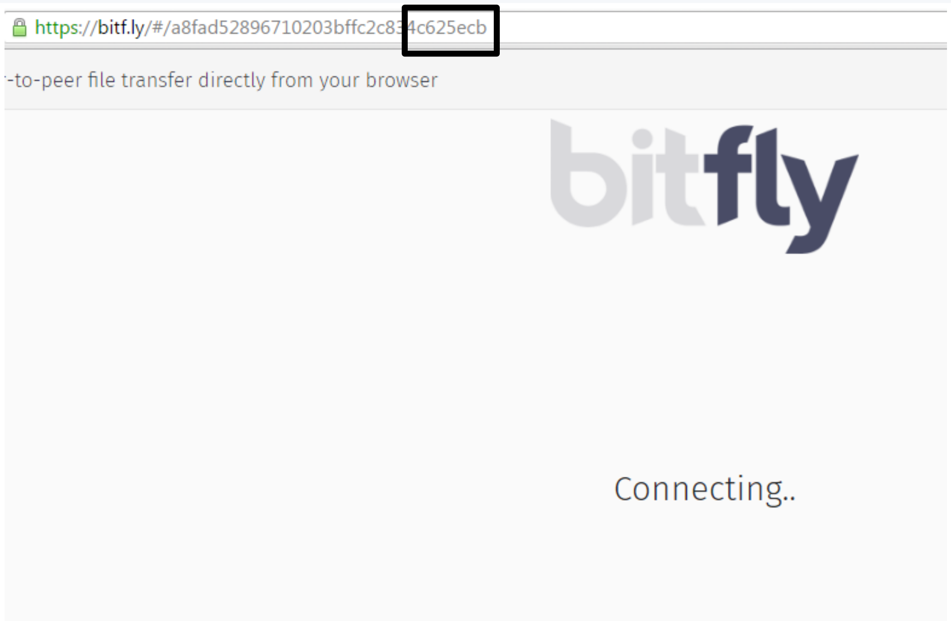
Open this URL on the receiving device

https://bitf.ly/#/a8fad52896710203bffc2c834c691ecb

and don't close this tab until the transfer is complete

FAQ

BITFLY HAS SHIPPED 7488 FILES SO FAR



https://bitf.ly/#/a8fad52896710203bffc2c834c625ecb

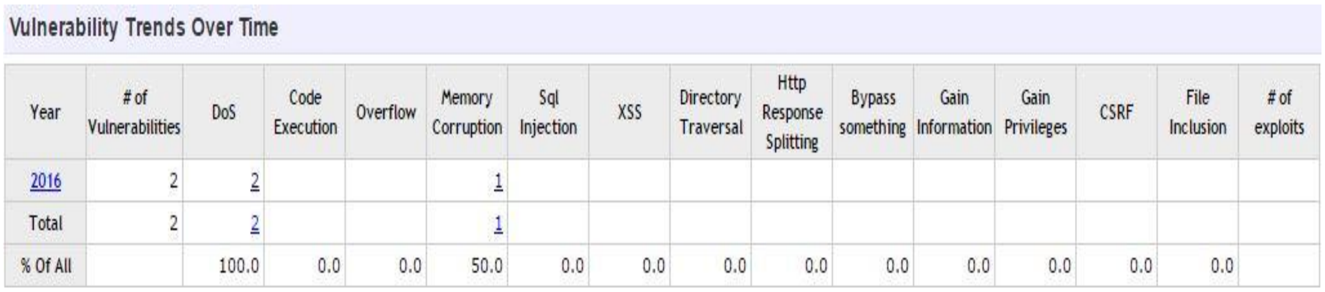
-to-peer file transfer directly from your browser

Connecting..

VULNERABILITIES:

1. WebRTC:

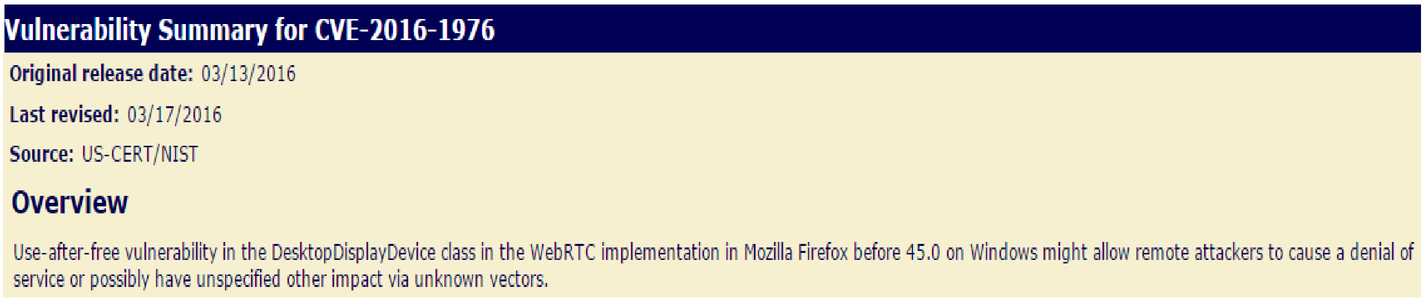
The Bitfly uses the webRTC for peer to peer file transfer. The webRTC has a lot of vulnerabilities as we will see in this section. I looked up webRTC on http://www.cvedetails.com/ website. It produced the following results.


Vulnerability Trends Over Time

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	

As we can see, 3 vulnerabilities have been reported in 2016. Out of these, 2 are denial of service attacks and 1 is a memory corruption attack. **[11]**

Description of one of the DOS attacks: [13]


Vulnerability Summary for CVE-2016-1976

Original release date: 03/13/2016

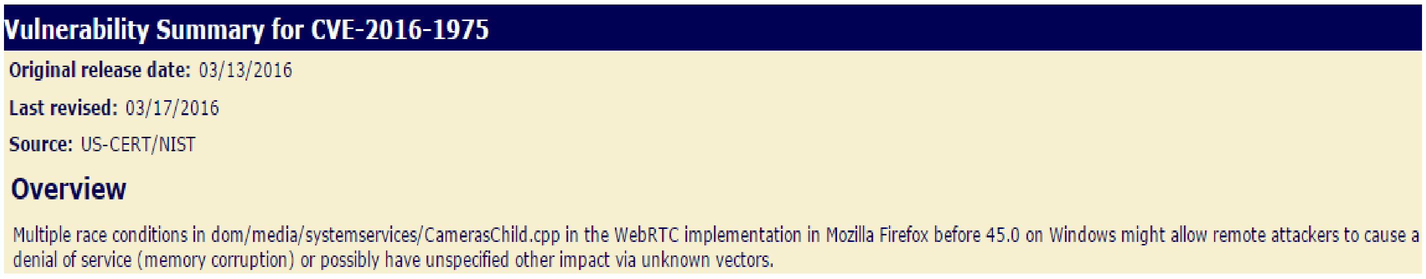
Last revised: 03/17/2016

Source: US-CERT/NIST

Overview

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.

Description of the other DOS attack and the memory corruption: [12]


Vulnerability Summary for CVE-2016-1975

Original release date: 03/13/2016

Last revised: 03/17/2016

Source: US-CERT/NIST

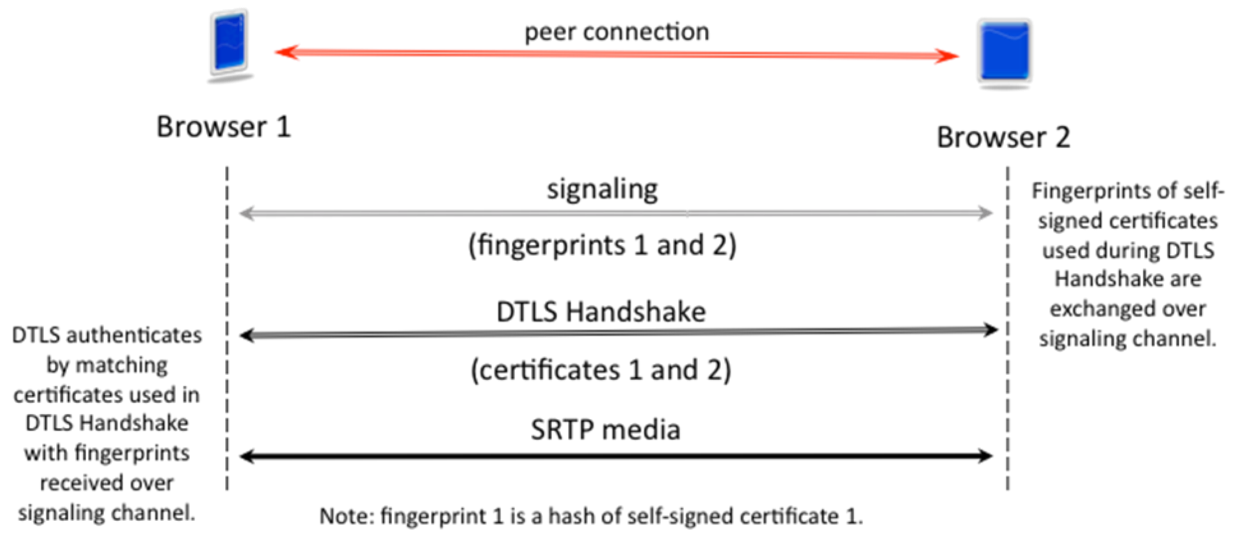
Overview

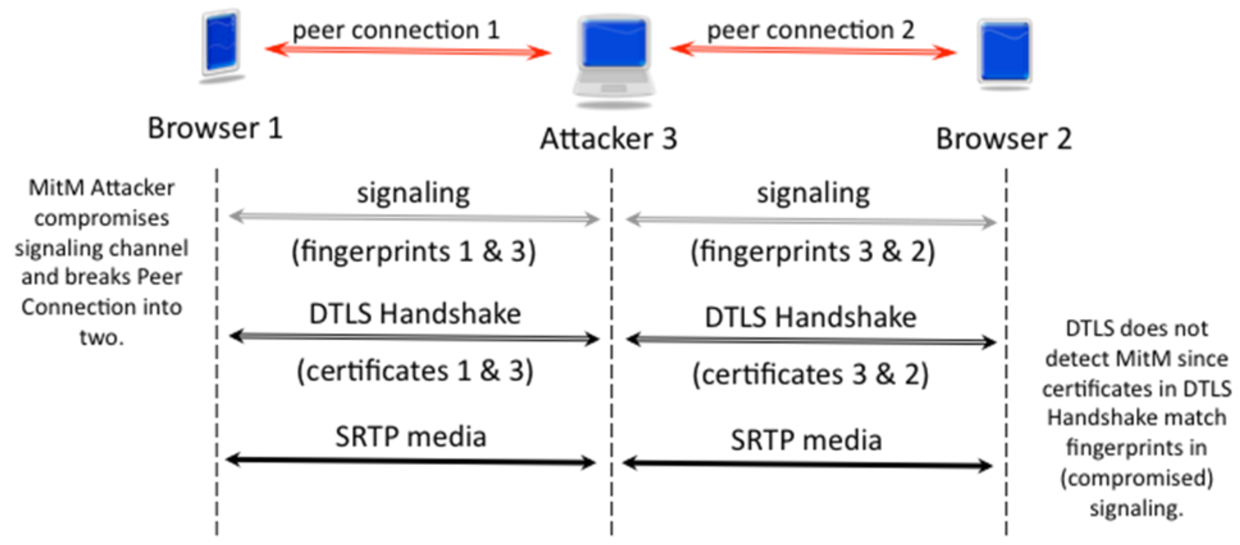
Multiple race conditions in dom/media/systemservices/CamerasChild.cpp in the WebRTC implementation in Mozilla Firefox before 45.0 on Windows might allow remote attackers to cause a denial of service (memory corruption) or possibly have unspecified other impact via unknown vectors.

- **WebRTC server: [7]**

The webRTC uses a server for signaling information which helps to setup the connection between the two communicating peers. This server can be hijacked or we can cause a denial of service attack by sending lots of signal traffic. Man-in-the-middle attacks are also possible.

Man-in-the-middle attack: [9]


peer connection
Browser 1
Browser 2
signaling
(fingerprints 1 and 2)
DTLS Handshake
(certificates 1 and 2)
SRTP media
Fingerprints of self-signed certificates used during DTLS Handshake are exchanged over signaling channel.
DTLS authenticates by matching certificates used in DTLS Handshake with fingerprints received over signaling channel.
Note: fingerprint 1 is a hash of self-signed certificate 1.



Note: Fingerprints refers to sharing of public keys between communicating parties.

Alan Johnston came up with this man-in-the-middle attack for webRTC. This was possible because the signaling information was not secured. He stated two reasons why performing a man-in-the-middle attack is very easy. The two reasons are as follows.

- ✓ It seems to be easy as well as quick to develop prototypes and test new ideas with WebRTC. Also, node.js seems to make it easy on the server side.
- ✓ WebRTC uses DTLS to produce keys for the SRTP media session but the normal protections offered by TLS are not present.

Description of the attack:

He had performed this attack as a demo while teaching a class on WebRTC at the Illinois Institute of Technology in Chicago. He had developed a demo application that uses an http polling signaling server that matches up two users that enter the same key and allows them to exchange SDP offers and answers. To perform the attack he compromised the server and made it work in such a way that whenever a user requests to be connected to another user, the user sending the request will be connected to the attacker and the requested user to whom the connection should be made to is also connected to the attacker. Therefore, this MITM attack forms two peer connections instead of one.

- **Fatal flaw - WebRTC exposes real IP addresses of VPN users: [8]**

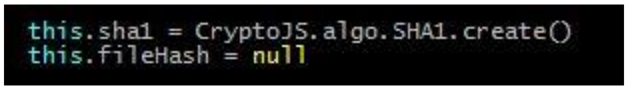
This fatal flaw enables the owner of a website to have access to the real IP address of users through WebRTC. A STUN (Session Traversal Utilities for NAT) server is used to provide the public IP address of an end host if it is located behind a NAT. WebRTC makes requests to the STUN server which in turn returns the hidden home IP address as well as local network address of the system used by the user. The results of the requests can be accessed using JavaScript, but because they are made outside the normal XML/HTTP request procedure, they are not visible in the developer console. There are mechanisms available to check if a system has been affected by the security glitch. Since chrome and firefox support WebRTC they have also taken measures to defend against this vulnerability. The chrome browser uses a WebRTC block or ScriptSafe extension to protect against this while Firefox uses NoScript extension that blocks javascript to tackle this. **The key thing to note here is that these extensions block the use of WebRTC or javascript and does not provide a means of using them safely.** There was another extension I found in chrome, WebRTC Leak Prevent. I installed this to see if it would let the application work normally. With this extension enabled the receiver's end is not able to make a connection in order to receive the file. Therefore, there is no proper solution yet available to enable safe WebRTC execution.

- **Possible effects on Bitfly as a result of using WebRTC:**

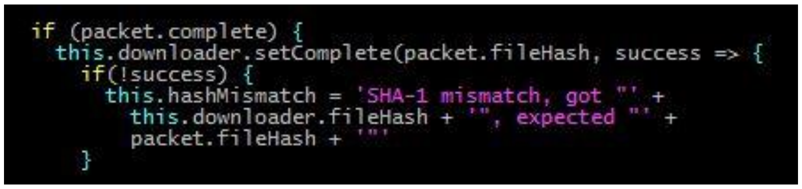
  - ✓ A denial of service attack is possible on the receiving system.
  - ✓ MITM allows a third party to receive the files. (Note: Since the URL is a one time use, a MITM will only allow the attacker to get the file)
  - ✓ Disclosure of the real IP address of the system is possible.
  - ✓ Compromising the WebRTC server is possible and hence can have the files sent to someone other than the intended person or do something else with the file.

2. File hashing:

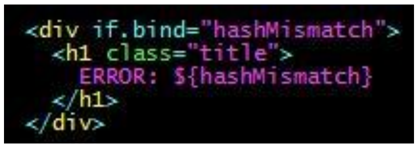
While surfing through the code I found that **SHA1** hash is being used on the file.



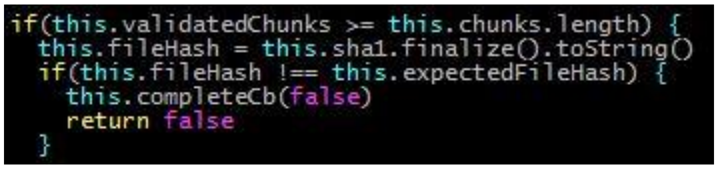
```
this.sha1 = CryptoJS.algo.SHA1.create()
this.fileHash = null
```



```
if (packet.complete) {
  this.downloader.setComplete(packet.fileHash, success => {
    if(!success) {
      this.hashMismatch = 'SHA-1 mismatch, got "' +
        this.downloader.fileHash + '", expected "' +
        packet.fileHash + '"'
    }
```



```
<div if.bind="hashMismatch">
  <h1 class="title">
    ERROR: ${hashMismatch}
  </h1>
</div>
```



```
if(this.validatedChunks >= this.chunks.length) {
  this.fileHash = this.sha1.finalize().toString()
  if(this.fileHash !== this.expectedFileHash) {
    this.completeCb(false)
    return false
  }
```

From the above code snippets we infer that the sha1 hash of the file is created to check the integrity of the file. Is SHA1 the right algorithm to use now? SHA-1 is indeed a good hashing algorithm. But recently it has been proved that calculating SHA-1 collisions is now feasible and easy **[3]**. Also, SHA-1 computations are slow **[5]**. In fact, as of 2016 many websites are no longer willing to support SHA-1 signatures or certificates **[4]**. Therefore, the developers should consider using other hashing algorithms.

3. **Getting the URL to the user:**

   The URL must be sent to the receiver by the sender. The sender can use any means to send the link. Some of the ways the user can do this include sending an email message, sending it through messenger options on social networking sites like facebook, using chatting applications such as whatsapp, mapping it to a QR code and sending the QR code and so on. This opens up the attack surface even further. All of these options have a variety of risks associated with them. Most of these can be performed on a mobile device as well or the URL can be sent as an SMS message increasing the attack surface even further. The file maybe sent securely but the means of sending the link to the receiver may be a weak one. For example, so many attacks are possible when sending it through email. A passive attacker can eavesdrop and get the link. Other attackers can do an MITM to get the URL. Therefore, the sender should be careful in choosing a proper medium to transmit the URL.

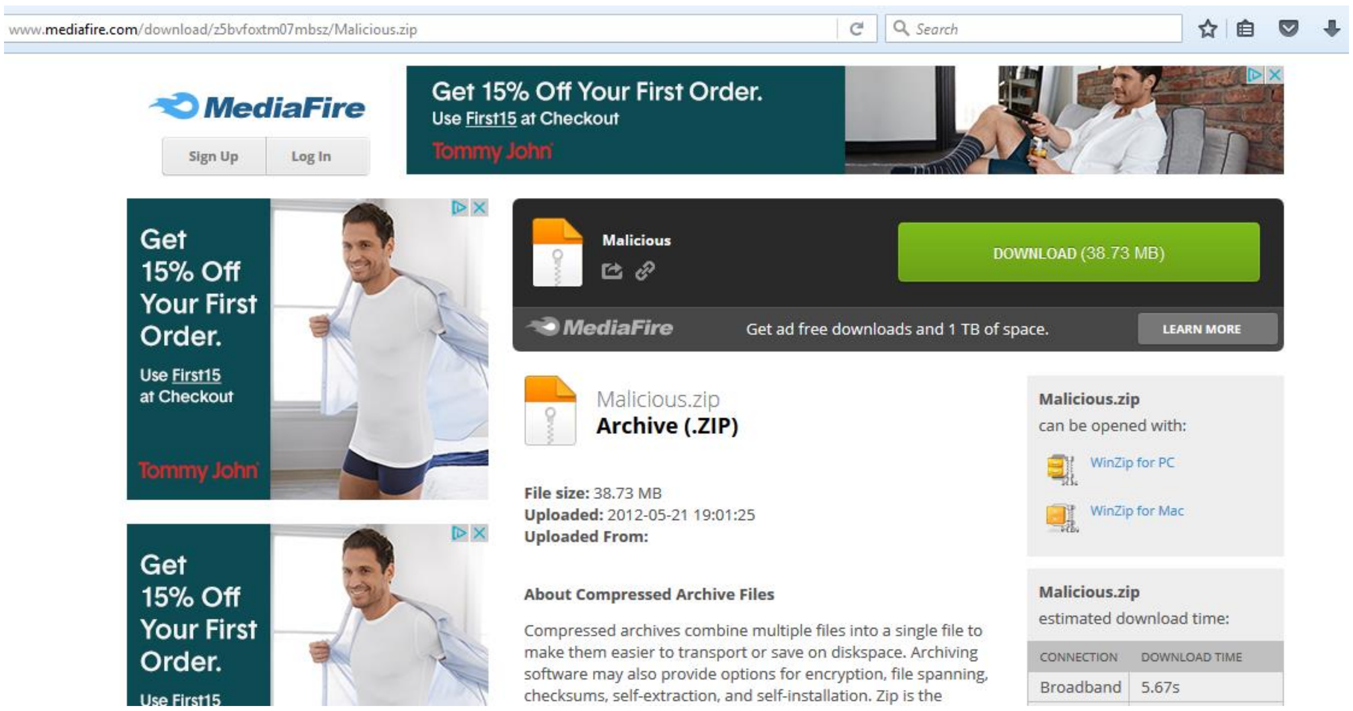
4. **Application can be misused to send malicious files to a user**:

   The application can be misused by someone to send malicious files to other users. The application does not parse any file that is being uploaded. It does not check the file for malicious code. Therefore, an attacker can use the application to compromise a particular system or in a different way to compromise more than one system. The attacker can compromise multiple systems. Use all these systems to obtain multiple URL's for the same file and send these URL's to a number of users. This could be considered as a type of spam attack. Other attacks are also possible.

Test: I have tested the application to see if it accepts a malicious file for transfer and as a result, have inferred if it checks files for malicious content.

The malicious file is downloaded from reference **[10]**.

Malicious file download:



www.mediafire.com/download/z5bvfoxtm07mbsz/Malicious.zip

MediaFire

Sign Up | Log In

Get 15% Off Your First Order.
Use First15 at Checkout
Tommy John

Malicious

DOWNLOAD (38.73 MB)

MediaFire Get ad free downloads and 1 TB of space. LEARN MORE

Malicious.zip
Archive (.ZIP)

File size: 38.73 MB
Uploaded: 2012-05-21 19:01:25
Uploaded From:

About Compressed Archive Files

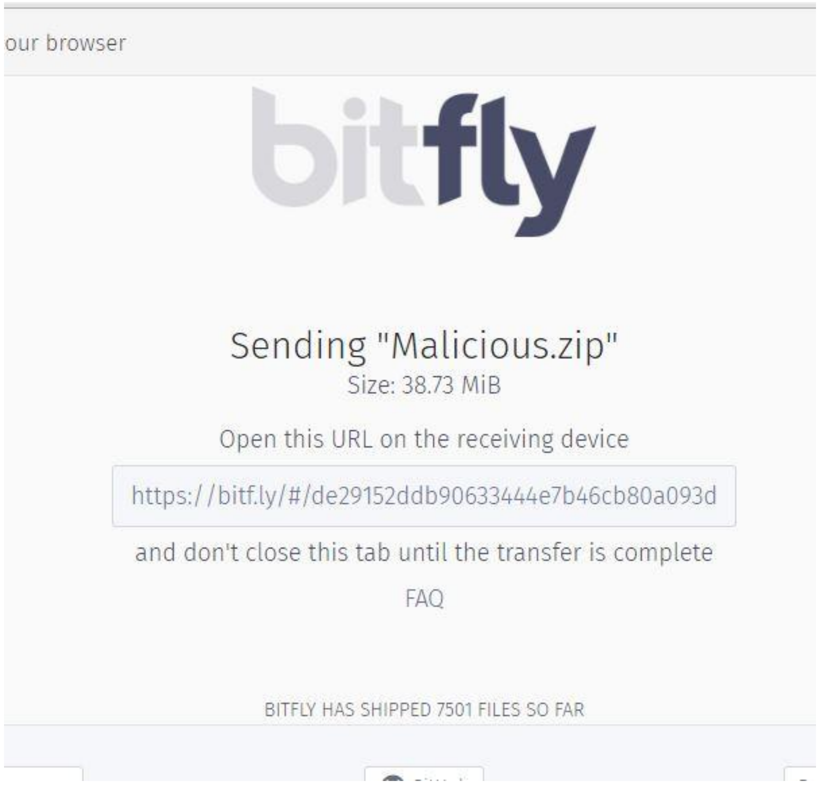
Compressed archives combine multiple files into a single file to make them easier to transport or save on diskspace. Archiving software may also provide options for encryption, file spanning, checksums, self-extraction, and self-installation. Zip is the

Malicious.zip
can be opened with:
WinZip for PC
WinZip for Mac

Malicious.zip
estimated download time:

CONNECTION	DOWNLOAD TIME
Broadband	5.67s

Sending malicious file:



our browser

bitfly

Sending "Malicious.zip"
Size: 38.73 MiB

Open this URL on the receiving device

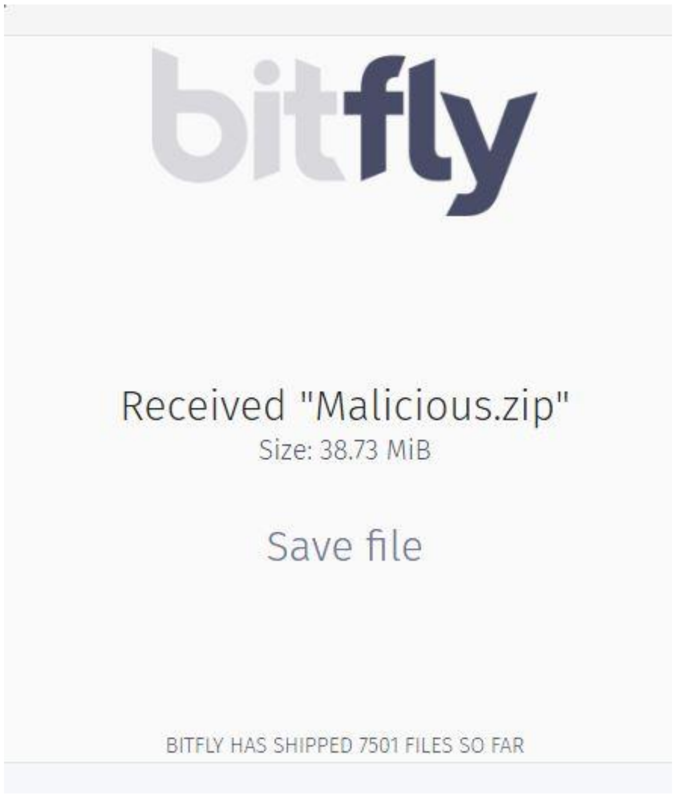
https://bitf.ly/#/de29152ddb90633444e7b46cb80a093d

and don't close this tab until the transfer is complete

FAQ

BITFLY HAS SHIPPED 7501 FILES SO FAR

Receiving malicious file:



Received "Malicious.zip"
Size: 38.73 MiB

PROPOSALS FOR IMPROVING THE APPLICATION:

- File hashing is done using a weak SHA-1 algorithm. This can be changed to a stronger algorithm like SHA-256.
- There doesn't to be an option to delete the uploaded file if a wrong file was chosen and send some other file instead. This feature should be incorporated as it is very important.
- Proper error messages should be displayed for each case. Currently, only some proper error messages have been implemented. In other cases like where the URL becomes invalid the browser continues to show the connecting window or redirects to the main page and does not display an error message.

FUTURE RESEARCH:

- Send multiple files each of size greater than 8 or 16 GB and see what happens.
- Analyze the ease of performing a man-in-the-middle attack and how it works.
- Analyze all the requests and responses to see if a passive attacker can obtain useful information that can give them access to the file.

CONCLUSION:

Overall this application provides a good means to transfer files securely. Many improvements are yet to be given before it can be released for use. As far as the security of files is concerned enough protection has been put in place as pointed out in this report. Some of the vulnerabilities do fall in the danger category and must be fixed before deployment.

REFERENCES:

1. https://bitf.ly
2. https://github.com/bitfly-p2p
3. https://en.wikipedia.org/wiki/SHA-1
4. https://blog.qualys.com/ssllabs/2014/09/09/sha1-deprecation-what-you-need-to-know
5. http://security.stackexchange.com/questions/61087/what-are-the-enhancements-of-sha1-compared-to-md5
6. https://bloggeek.me/send-file-webrtc-data-api/
7. http://blog.webrtc-solutions.com/webrtc-security-concerns/
8. http://thehackernews.com/2015/02/webrtc-leaks-vpn-ip-address.html
9. https://webrtchacks.com/webrtc-and-man-in-the-middle-attacks/
10. http://tradownload.com/results/malicious.html
11. http://www.cvedetails.com/product/33499/Webrtc-Project-Webrtc.html?vendor_id=15802
12. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1975
13. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1976