UNIVERSITY OF AMSTERDAM

Involuntary Browser-Based Torrenting

Alexander Bode abode@os3.nl

October 19, 2020

Abstract—**WebTorrent is the first torrent client to work in a browser. The web technology uses WebRTC for its peer-to-peer communications and is increasingly popular due to its variety of use cases. The rise in popularity has raised questions about WebTorrent's potential for abuse. This study aims to determine whether it is possible to use WebTorrent to have users involuntarily participate in peer-to-peer torrent networks. The WebTorrent API was analysed, and custom clients were built in order to determine if it is possible to run WebTorrent inconspicuously. The usefulness and attack vectors of involuntary browser-based torrenting have been analysed using a well-recognised security framework. Methods for the detection and prevention of WebTorrent usage have been investigated by inspecting the behaviour of the browser while using WebTorrent. A source-code search-engine was used to determine if this type of abuse is a widely used and established tactic. The results show that it is possible to use WebTorrent for involuntary browser-based torrenting and that it can be detected and prevented in several ways. Several proofs-of-concept are presented in this paper for the usage, detection and prevention of involuntary browser-based torrenting. Analysis has shown that this type of WebTorrent abuse is not an established tactic. This research has highlighted the importance of locking and requiring consent for the use of WebRTC interfaces and increases awareness of WebTorrent's potential for abuse.**

I. INTRODUCTION

Advances in web technologies have led to significant changes in how web servers can interact with web browsers. WebTorrent [1], a streaming torrent client, allows the browser to fetch and share content using distributed browser-to-browser networks. It is attracting considerable interest due to its variety of use cases. The WebTorrent protocol provides similar functionalities to that of the BitTorrent protocol but uses different data transport mechanisms. No additional browser plugins, extensions or installations are required for its use. It is completely written in JavaScript, a lightweight interpreted programming language and makes use of WebRTC [2], an open-source technology that supports browser-to-browser data transfers. The security implications of WebTorrent have not been established yet. Concerns have been raised which question its potential for abuse. This study aims to determine whether WebTorrent can be used for involuntary participation in browser-based peer-to-peer networks and evaluate the possibilities for an adversary that uses WebTorrent with malicious intent. Approaches for the detection and prevention of involuntary participation in WebTorrent networks, as well as searching for abuse in the wild are presented in this paper. Additionally, proofs-of-concept for the usage, detection and prevention of involuntary browser-based torrenting are presented in this paper.

RESEARCH QUESTIONS

The focus of this research is to determine whether it is possible to use WebTorrent for involuntary participation in peer-to-peer networks. The main research question is defined as the following:

Can WebTorrent be abused to have web page visitors involuntarily participate in peer-to-peer networks?

This question resulted in the following sub-questions:

- Which WebTorrent specific features can be abused?
- In which ways could WebTorrent be useful to an adversary?
- What can be done to prevent involuntary browser-based torrenting?
- Can we determine if this is an already established and used tactic?

A. Structure

The remainder of this paper is structured as follows. Section II reviews related work on WebRTC and WebTorrent. Section III provides context to the information that is discussed in this paper, which includes descriptions of BitTorrent, WebRTC and WebTorrent. The proposed approach for determining whether involuntary browser-based torrenting is possible is described in Section IV. Approaches for detection and prevention of WebTorrent usage, as well for determining if the abuse is an already established tactic are also given in this section. The findings of the experiments are shown in Section V. The key findings are summarised, and the implications and limitations of the results of Section V are discussed in Section VI. In Section VII, the research question is answered, and a conclusion is drawn based on the results of the experiments. Lastly, suggestions for future work are given in Section VIII.

II. RELATED WORK

Earlier work arises primarily from industry research and is presented in blog posts.

A. Web Real-Time Communication

The Internet Engineering Task Force has presented a document that defines the security architecture of WebRTC [3], the protocol suite that WebTorrent heavily relies on for real-time communication [4]. Detailed technical descriptions are given of possible security implications that are relevant to this study. HTML5Rocks has presented practical information on WebRTC's RTCDataChannel API [5], which is used by WebTorrent for direct browser-to-browser connections.

B. WebTorrent Evaluation

I. Koren and R. Klamma have implemented and evaluated a system that streams videos peer-to-peer via WebTorrent, named *OakStreaming* [6]. They have conducted tests with up to eight peers. Their primary motivations for the development were to reduce server loads and avoid transfers of intellectual property while maintaining a reasonable level of quality for its users.

Little academic research has been done to evaluate WebTorrent. In case WebTorrent can be used to have visitors involuntarily participate in browser-based peer-to-peer networks, this study will make efforts to find out how this can be done and what the implications may be.

III. Background

In this section, background information is given that is relevant to WebTorrent. We will take a look at the protocol and functionalities of WebTorrent that may be leveraged for browser-based torrenting.

A. BitTorrent

BitTorrent is a protocol that is used for distributing files using peer-to-peer connections. It is considered to be a peer-to-peer system, as file distribution services are both requested and provided amongst a set of endpoints [7]. A collection of endpoints that peer and share a specific file or set of files is known as a *swarm*. Files are split into parts, known as *pieces*, before distribution for improved efficiency. The protocol allows peers to download pieces while uploading other pieces that have already been completed. BitTorrent supports two kinds of peers, *seeders* and *leechers*. A peer that is both downloading and uploading in the swarm is known as a leecher. Peers that are only uploading and have all pieces of a specific resource are known as seeders. BitTorrent follows the *tit-for-tat* principle for distribution, i.e., if a peer wishes to receive data, it must also share data. The rarest pieces of a file are downloaded first. This way, the protocol creates an incentive for the peers to share data, which also helps preserve the availability of pieces in the long run. The protocol preempts slow downloads by searching for and adding additional peers. BitTorrent offers the possibility to ease the load on central servers, support a large number of downloaders and increase file distribution speeds.

BitTorrent's peer protocol runs over TCP or *μTP*; a transport protocol layered on top of UDP, known as the Micro Transport Protocol. The *μTP* protocol was later added as an extension to utilise unused bandwidth without disrupting internet connections [8].

BitTorrent file distribution consists of the following entities:

- BitTorrent client
- Static meta-info file
- Web server to host meta-info files
- BitTorrent tracker

To serve a single or a set of files, a meta-info file, also known as a *torrent*, can be created and published. The first BitTorrent client to upload a specific resource is known as the initial seeder. A torrent file contains information, such as the name and path of distributed files, how the file is split into pieces, file sizes and cryptographic hash values that are used for integrity checks [9]. Torrent files also contain data that can be used to find a swarm. Torrents are usually exchanged using a *torrent repository* web server.

Torrent files are used to establish connections to swarms. The meta-info files are used for both downloading and seeding. Participating in a swarm can be done using a BitTorrent client. The client may load a torrent file to identify servers that reveal the location of peers that can share a specific resource. BitTorrent clients are expected to keep uploading to the swarm after the download has completed. However, this is dependent on the client and its configuration [9]. A later extension to the BitTorrent specification allows clients to join a swarm without downloading a torrent file first. The meta-info is downloaded from peers instead. This method allows users to join a swarm using a *magnet link*. Magnet links identify files using cryptographic hash values, instead of locations [10].

BitTorrent *trackers* are servers that exist to assist in the communication between peers. A tracker maintains a list of endpoints that can share one or multiple pieces of a specific resource. It identifies a swarm and is only required during the initial establishment of the peer-to-peer connection [6]. Torrents often include a list of trackers. Alternatively, *trackerless torrents* use *Distributed Hash Tables* to store peer contact information. The support for trackerless torrents was later added as an extension to the BitTorrent specification. This feature allows every peer in the swarm to act as a tracker. BitTorrent clients include a *Distributed Hash Table node*, which acts as a listening client and server that implements the distributed hash table protocol. This node is used to contact other nodes in the table over UDP. Every node maintains a routing table of healthy nodes. Nonetheless, trackers can still be included in torrents to increase the speed of the discovery of peers in a swarm [11].

The creator of a torrent may specify fallback HTTP or FTP locations for files. These are uniform resource locators that are used to serve files using FTP or HTTP servers in addition to the peers in the swarm. This kind of fallback seeders are known as *WebSeeds* and are available as long as the server is available [12]. Torrents may include an optional list of HTTP servers.

B. WebRTC API

WebRTC, also known as *Web Real-Time Communication*, is an open-source software project that is supported by the majority of modern web browsers. The software allows adding real-time communication capabilities to web applications and supports browser-to-browser data transfers. It can be used for web applications that utilise camera's and microphones or for more advanced applications, such as the ones that support screen sharing [2]. WebRTC consists of several API's and protocols, which together support direct peer-to-peer connections, between one or multiple web browsers [6]. Connections can be established without installing additional software for the browser.

WebRTC implements three JavaScript API's:

- *MediaStream*
- *RTCPeerConnection*
- *RTCDataChannel*

MediaStream is used to represent a stream of audio or video and may contain multiple tracks. RTCDataChannel enables the peer-to-peer exchange of arbitrary data [5]. The RTCPeerConnection interface is used to represent connections between two peers, the local host and a remote host. Once the connection has been established, media streams or bi-directional data channels can be added, using the MediaStream and RTCDataChannel interfaces [13]. RTCPeerConnection shields developers from underlying complexities, such as packet loss concealment, echo cancellation, bandwidth adaptivity and dynamic jitter buffering. Applications that make use of the RTCPeerConnection API require a mechanism to coordinate communication. The mechanism is known as *signalling*. It is not specified by WebRTC but left up to the developer to implement. It is left undefined to maximise compatibility with existing technologies and to avoid redundancy [14].

Peer-to-peer connections are established using the Interactive *Connectivity Establishment (ICE)* protocol [15], which implements the *Session Traversal Utilities for NAT (STUN)* protocol [16] and its extension *Traversal Using Relays around NAT (TURN)* [17]. Initially, an attempt is made to connect peers directly over UDP. If this is not possible using UDP, then TCP is used instead. STUN servers are used to assist hosts in performing NAT traversal by responding with the public IP address and port of a client from the server's perspective. If direct peer to peer connections fail due to reasons, such as NAT and firewall constrictions, then TURN servers are used to relay data as a fallback. In this case, data is sent through a relay server, which consumes the server's bandwidth [18]. All WebRTC data streams are encrypted with the mandatory *Datagram Transport Layer Security* protocol, which is also known as *DTLS* [3].

C. WebTorrent

WebTorrent is a streaming torrent client, developed by Feross Aboukhadijeh. It is available for the browser and Node.js [1]. It is written in JavaScript, a lightweight interpreted programming language and is the first torrent client to work in a browser. In Node.js, it is a simple torrent client, which communicates using TCP and UDP. In the browser, it utilises WebRTC for its peer-to-peer transport and does not require any additional software installations. A great use case for WebTorrent is peer-assisted delivery, where users help host a website's content. The WebTorrent protocol in the browser is similar to the BitTorrent protocol except that it utilises WebRTC, instead of TCP or *μTP*, as its underlying transport protocol. Another difference is that WebTorrent has a custom implementation for tracker servers, which also serves to assist in the communication between peers. The developer claims that once peers are connected, WebTorrent will work the same as the BitTorrent protocol [4].

WebTorrent peers that utilise WebRTC, e.g., the peers that use WebTorrent in a browser, can only directly connect to other peers that utilise WebRTC. Peering with endpoints that use TCP, *μTP* or UDP is not supported in the browser. Hybrid torrent clients, such as *WebTorrent Desktop*, act as bridges between WebRTC WebTorrent peers and regular BitTorrent peers. These clients allow connecting to both BitTorrent and WebRTC WebTorrent peers [4].

IV. Methodology

In order to determine whether WebTorrent can be leveraged for involuntary participation in peer-to-peer networks, a local test environment was set up. The first part of the experiments focused on determining whether involuntary file-sharing with WebTorrent is possible, followed by an evaluation of its usefulness for a potential adversary. The second part was to determine methods for the detection and the prevention of WebTorrent usage. The third part focused on determining whether this kind of abuse with WebTorrent is an already established and widely used tactic.

A. Lab Setup

The test environment consisted of the following elements:

- A web server
- Multiple hosts running web browsers with support for JavaScript and WebRTC
- A JavaScript and WebRTC debugger on each host

The following operating systems were used to conduct the experiments:

- Kali GNU/Linux Rolling 2019.2 Virtual Machine
- Microsoft Windows 10 Pro Build 18362 Virtual Machine
- Mac OS Catalina 10.15.7 Machine
- Android 9 Mobile Device

The following web browsers were used to test the proof-of-concepts:

- Google Chrome 86.0.4240.75
- Mozilla Firefox 81.0.1
- Microsoft Edge 86.0.622.38
- Samsung Internet 12.1.4.3
- Opera 71.0.3770.228
- Safari 14.0

B. Involuntary File-Sharing with WebTorrent

The WebTorrent JavaScript methods and procedures that make it possible to do browser-to-browser file sharing were analysed. The analysis was performed by reading the official API documentation of WebTorrent [19] and using the Firefox browser's built-in JavaScript console and debugger. The local test environment was used to perform the experiments. In order to analyse the features and gain a better understanding of the WebTorrent library, two custom browser-based WebTorrent clients were written in JavaScript and HTML. The first client allowed seeding files, and the second client allowed downloading files. Each client was stored as a single HTML file, which contained the JavaScript WebTorrent code necessary to perform the experiments. In order to determine whether WebTorrent can be abused for involuntary file-sharing, proof-of-concept code was written and tested to see if it is feasible to perform involuntary browser-based torrenting inconspicuously. The open-source *Damn Vulnerable Web Application* software is an intentionally vulnerable PHP web application that exists to aid security professionals in testing their skills and tools in a legal environment [20]. It was used to determine if external payloads containing WebTorrent code could be successfully loaded and executed with cross-site scripting attacks.

The steps that were taken to determine the feasibility of involuntary browser-based torrenting can be outlined as follows:

1) Determine relevant methods of the API for simple WebTorrent usage
2) Write custom downloader and uploader clients
3) Debug and test custom client across various devices
4) Remove methods until the client can run inconspicuously in the browser
5) Write, debug and test involuntary browser-based torrenting proof-of-concepts

The usefulness of involuntary browser-based torrenting for an adversary was determined by analysing the attack vectors and the impact of the attack. In order to investigate if these attacks can be utilised with existing offensive techniques or as a part of a cyber-attack chain, the well-recognised *MITRE ATT&CK* framework was used as a guideline during the analysis. The matrix was created to be used for developing threat models and methodologies. MITRE has provided a matrix of adversary tactics and techniques based on real-world observations [21]. It is used for developing threat models and methodologies and was used in this study to help determine how involuntary browser-based torrenting could be used by an adversary, perhaps in a chained attack.

C. Detection and Prevention of WebTorrent

Methods for the detection and prevention of involuntary browser-based torrenting were determined by looking into the source code of WebTorrent, the behaviour of the browser and its underlying usage of JavaScript and WebRTC while running WebTorrent.

The Mozilla Firefox and Google Chrome browser both have built-in tools named *WebRTC Internals* that allow monitoring the statistics of active WebRTC sessions. These can be used for debugging applications that are built with WebRTC and can be accessed from within the browser using the URL's that are listed in Table I:

Browser	**URL**
Mozilla Firefox	chrome://webrtc-internals
Google Chrome	about:webrtc

TABLE I: WebRTC debuggers

The data was obtained using the WebRTC Internals tool of Google Chrome. The tool provides information, such as API traces and network properties, such as the transport protocols, STUN servers and TURN servers that are used by the ICE protocol during the establishment of the connection. As mentioned earlier in Section III, the RTCPeerConnection interface represents a WebRTC connection between the local host and a remote host. W3C has published a document that defines the *interface description language objects* that provide access to the statistical information about an RTCPeerConnection. These objects are returned from the *getStats API* [22], the application programming interface that both Google Chrome and Mozilla Firefox rely on to read the statistics of specific WebRTC connections. In order to interpret the output of Google Chrome's WebRTC Internals tool, third-party documentation supplied by TestRTC, together with the documentation by W3C were used [23].

Both browsers that were used during the experiments include built-in developer tools, including a JavaScript console, a JavaScript debugger and a network traffic monitor. These helped to understand what the code of the WebTorrent library did during execution. Pausing execution, controlling execution and inspecting values of variables and expressions, while monitoring the network traffic, was done to determine methods for the detection and prevention of WebTorrent use. Traffic inspection was also done using the popular network protocol analyser *Wireshark*. The tool has helped understand which protocols are involved during the use of WebTorrent in the browser. The source code was manually examined to take note of strings, classes and functions that are likely to be unique to WebTorrent. These were later used to create proofs-of-concept for the detection and prevention of involuntary browser-based torrenting and later also for searching for WebTorrent usage on the Internet.

D. Searching for Abuse in the Wild

To gain an impression of the popularity of the WebTorrent library and determine whether involuntary browser-based torrenting with WebTorrent is an established and widely used tactic, a source-code search-engine named *PublicWWW* [24] was used to search for pieces of code that are unique to the WebTorrent library. PublicWWW can find alphanumeric snippets, signatures or keywords in HTML, JS or CSS code. It supports using regular expressions to refine the search results and offers the option to exclude results containing a specific string. The search engine has indexed over a half-billion pages at the time of writing and allowed exporting results, sorted by page popularity, to a CSV file. The exported data was used for later analysis. Access to a professional account, one that allows searching the entire index, was on request kindly provided by PublicWWW's team.

V. Results

This section covers the outcomes of the performed experiments, including how involuntary browser-based torrenting can be done, an analysis of its usefulness for an adversary, proposed detection and prevention methods and methods for searching for abuse in the wild.

A. Involuntary File-Sharing with WebTorrent

The results of the experiments show that involuntary browser-based torrenting is possible using the WebTorrent library. The browser and the library do not prompt the user for permission to initiate peer-to-peer connections. The results include proofs-of-concept in the form of web documents, JavaScript files and a custom Mozilla Firefox browser extension. However, it is worth mentioning that torrents only remained actively uploading to the swarm as long as the victim kept the malicious web document open in the browser. This drawback does not apply to the browser extension proof-of-concept, which kept running until the browser was closed.

Involuntary browser-based torrenting was tested and successful on both desktop and mobile browsers listed in Table II.

Browser	Operating System	Result
Google Chrome 86.0.4240.75	Android, Linux, Windows	Vulnerable
Mozilla Firefox 81.0.1	Android, Linux, Windows	Vulnerable
Microsoft Edge 86.0.622.38	Windows	Vulnerable
Samsung Internet 12.1.4.3	Android	Vulnerable
Opera 71.0.3770.228	Linux	Vulnerable
Safari 14.0	Mac OS	Vulnerable

TABLE II: Tested web browsers

The first step towards developing a proof-of-concept involved taking note of the JavaScript methods that were used for building the custom *WebTorrent Downloader* and the *WebTorrent Uploader* clients [25]. The custom WebTorrent Downloader and Uploader clients can be retrieved from the GitHub page that has been set up for this research [26]. The JavaScript methods that were used can be seen in Table III.

The second step involved removing code from the custom clients until each client only used JavaScript methods that are required to download or seed files. This step resulted in two proofs-of-concept, HTML web documents, each containing HTML and JavaScript code. The proofs-of-concept demonstrate that it is possible to use the WebTorrent library without the consent of an end-user. The *Stealth Downloader* proof-of-concept includes an optional `file.getBlobURL` JavaScript method to store the location of the downloaded resource to a variable. The value could be retrieved using the JavaScript console and the browser's *Window* interface.

JavaScript Method	Purpose
new WebTorrent([options])	Create new WebTorrent client instance
client.seed(file, [callback])	Create torrent and seed file
client.add(torrent, [callback])	Add torrent and start downloading
file.getBlobURL	Return local URL to (cached) file
torrent.magnetURI	Return Magnet URI of torrent
torrent.torrentFileBlobURL	Return local URL to (cached) torrent file
torrent.ratio	Return download/upload ratio
torrent.numPeers	Return number of connected peers
torrent.wires	Return addresses of connected peers
torrent.downloadSpeed	Return current download speed
torrent.uploadSpeed	Return current upload speed
torrent.progress	Return torrent progress

TABLE III: WebTorrent JavaScript methods

The Window interface represents a window containing the *Document Object Model* [27]. To further clarify, this means that retrieving the value of the stored variable makes it possible to download the resource from the browser's cache to a location of choice. It is important to note that browsers have limited cache available for WebTorrent downloads. The available cache storage differs from browser to browser. The proof-of-concept Stealth Downloader can be retrieved from the GitHub page that has been set up for this research [26]. The WebTorrent JavaScript methods that were used for this proof-of-concept can be seen in Table IV.

JavaScript Method	Purpose
new WebTorrent([options])	Create new WebTorrent client instance
client.add(torrent, [callback])	Add torrent and start downloading
file.getBlobURL	Return local URL to file

TABLE IV: Stealth Downloader JavaScript methods

The *Stealth Seeder* proof-of-concept includes a custom function to convert a JavaScript byte-array to a file, which is then used as input for the `client.seed` JavaScript method. It also includes two WebTorrent JavaScript methods, which are used to store the magnet URI and the local torrent file URL in a variable. This can be read using the JavaScript console and the Window interface, in the same way as with the Stealth Downloader. The *Background Seeder* proof-of-concept, a Firefox browser extension, requires that an initial seeder shares a resource first so that it can be downloaded and shared once again by the Background Seeder. The use of a WebSeed is recommended in this case. The WebTorrent Background Seeder extension can be retrieved from the GitHub page that has been set up for this research [26]. The WebTorrent JavaScript methods that were used for the Stealth Seeder can be seen in Table V.

JavaScript Method	Purpose
new WebTorrent([options])	Create new WebTorrent client instance
client.seed(file, [callback])	Create torrent and seed file
torrent.magnetURI	Return Magnet URI of torrent
torrent.torrentFileBlobURL	Return local URL to (cached) torrent file

TABLE V: Stealth Seeder JavaScript methods

Multiple JavaScript proofs-of-concept were written. The WebTorrent JavaScript methods used in the JavaScript proofs-of-concept are identical to the ones used in the Stealth Downloader and Stealth Seeder proofs-of-concept, except for a single custom function that is required for cross-site scripting attacks. The results of the experiments with the Damn Vulnerable Web Application software show that cross-site scripting attacks that load external JavaScript files containing WebTorrent code is possible. A requirement for the attack to be successful was to implement a custom function that waits for the page and its resources to finish loading, before loading the WebTorrent library and executing the code responsible for downloading or seeding. The custom function loads the WebTorrent library from an external resource by appending a JavaScript script source tag in the head section of the document. It continues by executing WebTorrent code, which is implemented as a callback function. The JavaScript proofs-of-concept can be retrieved from the GitHub page that has been set up for this research [26].

An adversary may use the following attack vectors to reach victims for involuntary browser-based torrenting:

- Malicious or compromised web servers
- Compromised externally hosted JavaScript libraries
- Malicious browser extensions

An adversary could set up a web page and run WebTorrent code in the background without the user's consent or use a compromised web server. Compromising an externally hosted JavaScript library has the benefit of reaching all web servers that include the library and thus its clients. A browser extension that downloads and seeds a specific torrent in the background was written for Mozilla Firefox, in addition to the web document and JavaScript proofs-of-concept. Creating the background seeder was done to demonstrate that it is possible to do involuntary browser-based torrenting using a browser extension.

An adversary may use involuntary browser-based torrenting for the following:

- Resource hijacking
- Repudiation
- Data exfiltration

An attacker could let a victim involuntarily participate in swarms to increase the download speed of a particular resource. This kind of attack is known as resource hijacking. Increasing the number of users will generally increase the availability of torrent pieces. The protocol makes sure that the rarest piece is downloaded first, which is beneficial for the health of torrents in the long run. Another use case is repudiation. As a number of clients are forced to download a specific resource, it becomes harder to determine who intended to receive the downloaded data. It will not be visible on the server-side if a client retrieves the file from a local cache using the WebTorrent's `file.getBlobURL` JavaScript method. WebTorrent may also be useful for data exfiltration, as all communications have mandatory Data Transport Layer Security encryption, which makes it harder to distinguish benign traffic from malicious traffic. DTLS is often used to encrypt benign traffic, such as that of voice calls or video conferences. Nonetheless, it might be useful for an adversary in certain cases.

B. Detection and Prevention

Further analysis has helped to determine the behaviour of WebTorrent that could be leveraged for detection and prevention methods. Observations are listed in Table VI.

Level	**Purpose**
Browser	Window interface is loaded with WebTorrent objects
Browser	WebSocket handshake may be performed with trackers
Browser	WebTorrent library may be included with common name
Browser	WebRTC communication exists with STUN servers
Network	DNS queries are done for torrent trackers
Network	DNS queries are done for STUN servers

TABLE VI: WebTorrent behaviour

1) Window Interface: The Window interface allows reading JavaScript variables that are currently loaded in the Document Object Model. This interface was leveraged to detect common names of loaded WebTorrent JavaScript objects, such as "WebTorrent". A custom Mozilla Firefox browser extension was written that attempts to detect and block WebTorrent usage by checking the Window for specific names and by prompting the user for permission to use WebTorrent. This kind of extension is known as a *content script* [28]. The *WebTorrent Blocker* browser extension runs the WebTorrent JavaScript method `client.destroy()` when WebTorrent usage has been rejected through the prompt. `The client.destroy()` method prevents the client from starting by destroying the client before peer-to-peer connections take place. The browser extension relies on the `wrappedJSObject` JavaScript property, which allows sharing JavaScript objects with browser extensions. Web pages usually do not expose loaded JavaScript objects to browser extensions, as this might have security implications. It is good security practice to rewrap objects with the `XPCNativeWrapper()` JavaScript method, once they have been read with the `wrappedJSObject` property [29]. The code responsible for detecting names of WebTorrent objects can be seen in Appendix IX-D. The WebTorrent Blocker proof-of-concept browser extension can be retrieved from the GitHub page that has been set up for this research [26].

2) WebSocket Handshake: WebTorrent clients communicate with trackers in order to establish connections with swarms. This is done using the WebSocket protocol. Initially, the WebSocket server listens using a standard TCP socket on a web server. To establish a WebSocket connection, an opening handshake, an HTTP Upgrade request, is sent by the client. Both parties negotiate the details of the connection, and the connection is established upon a successful negotiation [30]. The opening handshake request may contain the URL of the tracker; an example can be seen in Appendix IX-A. The handshake may exhibit connections with trackers but is usually encrypted with TLS. These requests can be blocked by blacklisting the WebSocket URLs to the torrent trackers, using a browser extension, such as *uBlock Origin*. The uBlock Origin browser extension allows implementing a static filter list. The extension uses a custom filter syntax [31]. The results showed that it is possible to block torrents that require a tracker server to find its peers. This solution does not apply to torrents that make use of Distributed Hash Tables. The static filter list used in this experiment can be seen in Appendix IX-C. Blocking the trackers might prevent peers from finding the swarm, thus making it unavailable for file-sharing. A list of default trackers that WebTorrent uses was retrieved from the library's source code.

3) Included WebTorrent Library: The WebTorrent library may be included using the JavaScript source attribute. These includes can be detected and prevented in two ways. The first option is to check for the common names of the WebTorrent library, *webtorrent.min.js* or *webtorrent.js*, then block URLs containing the common name. Blocking URL's can be done using a browser extension, such as uBlock Origin. The second option is to filter all web browser responses that contain JavaScript files with patterns that are unique to the WebTorrent library. A custom Mozilla Firefox browser extension was written, which relied on Firefox's `browser.webRequest.filterResponseData()` JavaScript method to allow monitoring and modifying requests in transit [32]. The filter drops requests upon detecting specific strings. It is worth mentioning that page loading times are increased and that web pages may not load correctly due to a currently present bug in the API [33]. Making the library inaccessible could prevent WebTorrent clients from being created. The *WebTorrent Filter* proof-of-concept browser extension can be retrieved from the GitHub page that has been set up for this research [26].

4) WebRTC Signalling: STUN servers allow WebTorrent clients to perform NAT traversal by returning the public IP address and port of a client from the server's point of view, as previously mentioned in Section III. This behaviour is part of WebRTC's signalling process. WebTorrent includes public STUN servers in its source code, which can be seen in Appendix IX-E. Access to the IP addresses of these STUN servers could be blocked on a network level, e.g., in the network router. Blocking STUN servers could prevent WebRTC from establishing peer-to-peer connections for WebTorrent, as fallback TURN servers have not been included in the WebTorrent library and need to be added manually as an option when creating a new WebTorrent client instance. A summary of a packet capture that shows a STUN binding request and response with the default server and the client is shown in Appendix IX-F. It is worth mentioning that Google's STUN server is public, blocking it might cause other benign web applications to stop working.

5) DNS Requests: The web browser performs DNS requests to discover the IP addresses of torrent trackers and STUN servers when a WebTorrent client is created. Blocking DNS requests for these specific domain names might prevent a WebTorrent client from participating in a swarm, as these might be required for initiating connections. Summaries of the packet capture that show DNS queries for the STUN servers and trackers can be seen in Appendix IX-G and IX-H respectively.

6) WebRTC Interface Locking: A more robust and elegant solution would be to prompt the user and ask for permission to use the RTCPeerConnection WebRTC interface. The `MediaDevices.getUserMedia()` JavaScript method prompts the user for permission to unlock access to a microphone or camera [34]. A similar mechanism could be implemented for the RTCPeerConnection interface, which would alert the user when peer-to-peer connections are initiated and prevent these when permission is not granted.

C. Searching for Abuse in the Wild

The results have shown that involuntary browser-based torrenting is not a widely used and established tactic. The source-code search-engine PublicWWW was used to search for WebTorrent script includes, JavaScript methods, base64 encoded strings and hexadecimal encoded strings. The common name of the WebTorrent library webtorrent.min.js resulted in 307 listed pages, which was the most extensive result and helped estimate the popularity of the library. Searching for the new `WebTorrent()` JavaScript method returned 50 results, of which all pages were manually examined and found to be benign WebTorrent usage or a false positive. Searching for base64 or hex-encoded patterns did not yield any results. The search queries that were used can be seen in Appendix IX-I. The CSV exports of the search results can be retrieved from the GitHub page that has been set up for this research [26].

VI. Discussion

The overall purpose of this research is to determine whether it is possible to leverage WebTorrent for having users participate in peer-to-peer networks involuntarily. The results indicate that involuntary browser-based torrenting is possible and that user consent is not a requirement for initiating peer-to-peer connections. Both the WebTorrent library and the browser fail to prompt the user, which makes it possible to have the user create and run a WebTorrent client inconspicuously in the background. Malicious WebTorrent code could reach users through web pages, browser extensions or externally hosted JavaScript library includes. Moreover, WebTorrent could be used for resource hijacking, the repudiation of downloads and data exfiltration.

Several methods have been presented for the detection and prevention of WebTorrent usage. The most remarkable result that emerged from these methods is the WebTorrent Blocker browser extension. The approach for this proof-of-concept is likely applicable to other JavaScript libraries. Nonetheless, a more robust solution would be to implement a permission-based locking mechanism for the WebRTC interface that is responsible for peer-to-peer connections. This solution is the most reliable way to detect and prevent WebTorrent usage, as WebTorrent relies on peer-to-peer connections for its distribution of data. No significant WebTorrent abuse was found by analysing the results of the source-code search-engine. The results, therefore, suggest that involuntary browser-based torrenting is not a widely used and established tactic. The findings of this study are well substantiated by factual information, which can be found on the project's GitHub page. Reproducing the experiments delivers consistent results. As far as we know, this is the first time that the potential abuse of WebTorrent is researched. This study provides considerable insight into involuntary browser-based file-sharing with WebTorrent.

Given that the findings, which determined whether involuntary browser-based torrenting is a widely used and established tactic, are based on a single source-code search-engine, the results from such analyses should consequently be treated with considerable caution. We are aware that the proofs-of-concept that are presented in this paper have limitations. Inconspicuously running WebTorrent clients are destroyed when the browser or malicious page is closed, or the malicious page is refreshed. It is beyond the scope of this study to address the question of how persistence can be achieved. Furthermore, the WebTorrent blocker extension depends on finding common names of WebTorrent objects, which could be altered in order to circumvent detection and prevention.

VII. Conclusion

This research aimed to determine whether WebTorrent can be abused to have web page visitors involuntarily participate in peer-to-peer networks. The results of this study clearly indicate that involuntary browser-based torrenting is possible and highlight the importance of requiring consent for the use of WebRTC interfaces. Based on a qualitative and experimental analysis, it can be concluded that involuntary browser-based torrenting can be detected and prevented in various ways. It is useful for resource hijacking, data exfiltration and repudiation but is not a widely used and established tactic. Following the proposed methodology, several proofs-of-concept were created and presented in this paper, including stealth WebTorrent clients and browser extensions that detect and block WebTorrent usage. However, a better solution would be to implement a permission-based locking mechanism for WebRTC interfaces that are used by WebTorrent. Furthermore, an important limitation lies in the fact that a single source-code search-engine was used to determine if it is a widely used and established tactic. Nonetheless, we believe that the results of this study could increase awareness of its potential for abuse.

VIII. Future work

Currently, there are no known methods for maintaining the state of torrents across browsing sessions when using WebTorrent. To further clarify, even though it is possible to do involuntary browser-based torrenting, the usefulness limited due to the volatility of the WebTorrent client in the browser. Further work needs to be done to determine methods for achieving persistence. Further experimental tests are needed to determine if involuntary browser-based torrenting can be used in other ways than presented in this paper. On a wider level, research is also needed to determine the feasibility of real-world attacks. Another important matter to resolve in future studies is to find a way to address the absence of user consent when certain WebRTC interfaces are used.

References

[1] Webtorrent. [Online]. Available: https://webtorrent.io/

[2] Webrtc - real-time communication for the web. [Online]. Available: https://webrtc.org/

[3] E. Rescorla, "Ietf - webrtc security architecture," *Work in Progress*, 2013. [Online]. Available: https://ftp.ripe.net/rfc/v3test/testing_dl_v25.pdf

[4] Webtorrent faq. [Online]. Available: https://webtorrent.io/faq/

[5] D. Ristic. Webrtc's rtcdatachannel api. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/datachannels/

[6] I. Koren and R. Klamma, "Peer-to-peer video streaming in html5 with webtorrent," in *International Conference on Web Engineering*. Springer, 2018, pp. 404–419.

[7] Rfc#5694: Peer-to-peer (p2p) architecture. [Online]. Available: https://tools.ietf.org/html/rfc5694#section-2.4

[8] Bep#29: utorrent transport protocol. [Online]. Available: http://bittorrent.org/beps/bep_0029.html

[9] Bep#3: The bittorrent protocol specification. [Online]. Available: http://bittorrent.org/beps/bep_0003.html

[10] Bep#9: Extension for peers to send metadata files. [Online]. Available: http://bittorrent.org/beps/bep_0009.html

[11] Bep#5: Dht protocol. [Online]. Available: http://www.bittorrent.org/beps/bep_0005.html

[12] Bep#19: Webseed - http/ftp seeding. [Online]. Available: http://bittorrent.org/beps/bep_0019.html

[13] Mdn web docs: Webrtc api. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

[14] Getting started with webrtc. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/basics/

[15] Rfc#8445 - interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal. [Online]. Available: https://tools.ietf.org/html/rfc8445

[16] Rfc#5389 - session traversal utilities for nat (stun). [Online]. Available: https://tools.ietf.org/html/rfc5389

[17] Rfc#5766 - traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). [Online]. Available: https://tools.ietf.org/html/rfc5766

[18] Webrtc in the real world: Stun, turn and signaling. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

[19] Webtorrent documentation. [Online]. Available: https://webtorrent.io/docs/

[20] Damn vulnerable web application. [Online]. Available: https://github.com/digininja/DVWA

[21] Mitre attck. [Online]. Available: https://attack.mitre.org/

[22] W3c - identifiers for webrtc's statistics api. [Online]. Available: https://www.w3.org/TR/webrtc-stats/

[23] The missing chrome://webrtc-internals documentation. [Online]. Available: https://testrtc.com/webrtc-internals-documentation/

[24] Publicwww: Search engine for source code. [Online]. Available: https://publicwww.com/

[25] Custom webtorrent clients. [Online]. Available: https://github.com/alexander-47u/Involuntary-WebTorrent-Test/tree/main/1-webtorrent-test-clients/custom-clients

[26] Github project page: Involuntary webtorrent test. [Online]. Available: https://github.com/alexander-47u/Involuntary-WebTorrent-Test

[27] Mdn web docs: Window interface. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window

[28] Mdn web docs: Content scripts. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts

[29] Mdn web docs: wrappedjsobject. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Sharing_objects_with_page_scripts

[30] Rfc#6455 - the websocket protocol. [Online]. Available: https://tools.ietf.org/html/rfc6455#section-1.3

[31] ublock origin browser extension. [Online]. Available: https://github.com/gorhill/uBlock

[32] Mdn web docs: webrequest.filterresponsedata() api. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/filterResponseData

[33] webrequest.filterresponsedata() bug. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=1561604

[34] Mdn web docs: Mediadevices.getusermedia() method. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

IX. Appendix

A. WebSocket Upgrade Request

```
Host: tracker.openwebtorrent.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Sec-WebSocket-Version: 13
Origin: null
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: 9JTwGTG/y3zs/+o7iL3dDw==
DNT: 1
Connection: keep-alive, Upgrade
Cookie: __cf_bm=07a422fa5ec0fbb068206aa734b3ad1b809a2c0f-1602169183-1800-Aat0hH+pWz3DDzZNIrfE⌋
↪  NBZupxwI8VPjGNgzbu7y/ufuKNmHoagW+/bimGH0up/qrEgexJ5XD2C/H8KNokedeSM=
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

B. WebTorrent Filter JavaScript Code

```
// WebTorrent Filter

// Array of strings to filter
var detectionStrings = ['WEBTORRENT_ANNOUNCE']

// Function for creating a HTTP filter
function listener(details) {

  // Create stream filter object
  let filter = browser.webRequest.filterResponseData(details.requestId);

  // Create text encoder/decoder objects
  let decoder = new TextDecoder("utf-8");
  let encoder = new TextEncoder();

  // If filter detects strings, drop response
  filter.ondata = event => {
    let str = decoder.decode(event.data, {stream: true});
        if (contains(str, detectionStrings)){
                filter.close()
        }
        else {
 // If filter does not detect string, pass response and disconnect filter
            filter.disconnect();}
  }

  return {};
}

// Function for checking for specific string
function contains(target, pattern){
    var value = 0;
    pattern.forEach(function(word){
      value = value + target.includes(word);
    });
    return (value === 1)
}

// Listener function executes when a request is about to be made, and before headers are
↪  available
browser.webRequest.onBeforeRequest.addListener(
  listener,
  {urls: ["<all_urls>"], types: ["script"]},
  ["blocking"]
);
```

C. uBlock Origin Static Filter List for WebTorrent

```
udp://tracker.leechers-paradise.org:6969
udp://tracker.coppersurfer.tk:6969
udp://tracker.opentrackr.org:1337
udp://explodie.org:6969
udp://tracker.empire-js.us:1337
wss://tracker.btorrent.xyz
||tracker.btorrent.xyz^$websocket,domain=file-scheme,important
wss://tracker.openwebtorrent.com
https://cdn.jsdelivr.net/npm/webtorrent@latest/webtorrent.min.js
```

D. WebTorrent Blocker JavaScript Code

```
// Checks if typical names of WebTorrent objects are in object window using "wrappedJSObject"
if ((typeof window.wrappedJSObject.WebTorrent != "undefined") || (typeof
↪  window.wrappedJSObject.client != "undefined")) {
          // Prompts user to ask if WebTorrent user is allowed
          if (confirm("WebTorrent Usage Detected! \nAllow file sharing?")) {
                          // Do nothing!
    }
      else {
                        // Close WebTorrent client using WebTorrent's destroy() function
                        if (window.wrappedJSObject.WebTorrent){
                                window.wrappedJSObject.WebTorrent = {};
                                        }
                        if (window.wrappedJSObject.cl){
                                window.wrappedJSObject.cl.destroy();
                                        }
                        if (window.wrappedJSObject.client){
                                window.wrappedJSObject.client.destroy();
                                        }
                  }
}

// Rewrap object since unwrapping is transitive
XPCNativeWrapper(window.wrappedJSObject.WebTorrent);
```

E. WebTorrent Hardcoded STUN Server from Source-Code

```
"stun:stun.l.google.com:19302"
stun:global.stun.twilio.com:3478?transport=udp
```

F. WireShark Capture: STUN Binding

```
No.     Time                Source           Destination      Protocol    Length  Info
58      1.066393439         172.16.217.136   74.125.128.127   STUN        62      Binding
↪  Request

No.     Time                Source           Destination      Protocol    Length  Info
59      1.079101737         74.125.128.127   172.16.217.136   STUN        74      Binding
↪  Success Response XOR-MAPPED-ADDRESS:[REDACTED PUBLIC IP]
```

G. WireShark Capture: DNS Query for STUN Servers

```
No. Time            Source          Destination     Protocol    Length  Info
308 6.133504005     192.168.192.4   216.239.32.10   DNS         88      Standard query 0xc768
↪  A stun.l.google.com OPT

No. Time            Source          Destination     Protocol    Length  Info
309 6.133526348     192.168.192.4   84.53.139.64    DNS         93      Standard query 0x7dd2
↪  A global.stun.twilio.com OPT
```

H. WireShark Capture: DNS Query for Tracker

```
No. Time              Source           Destination      Protocol    Length  Info
11  0.847021744       172.16.217.136   172.16.217.2     DNS         80      Standard query 0x2e4e
 ↪  A tracker.btorrent.xyz

No. Time              Source           Destination      Protocol    Length  Info
12  0.847108950       172.16.217.136   172.16.217.2     DNS         80      Standard query 0x9f4a
 ↪  AAAA tracker.btorrent.xyz
```

I. PublicWWW Search Results

```
Description                  | Results | Exact Query
=========================================================================================================================
WebTorrent library           | 308     | "webtorrent.min.js" depth:all

new WebTorrent() method      | 50      | "new WebTorrent();" depth:all

client.add() method          | 86      | "webtorrent.min.js" snipexp:|WebTorrent\(\);.*?\.add\(| depth:all

client.add() method          | 15      | "new WebTorrent" snipexp:|WebTorrent\(\);.*?\.add\(| depth:all

hex string "webtorrent"      | 0       | "\\x77\\x65\\x62\\x74\\x6f\\x72\\x72\\x65\\x6e\\x74" depth:all

hex string "WebTorrent"      | 0       | "\\x57\\x65\\x62\\x54\\x6f\\x72\\x72\\x65\\x6e\\x74" depth:all

hex string "magnet:"         | 0       | "bWFnbmV0Oj" depth:all

base64 string "webtorren"    | 0       | "d2VidG9ycmVud" depth:all

base64 string "WebTorren"    | 0       | "V2ViVG9ycmVud" depth:all

base64 string "magnet:"      | 0       | "bWFnbmV0Oj" depth:all
```