Please Ensure You Are Seeing the Entire View of The PowerPoint,

Some Issues occur with Zoom's Default View.

Beginning

Securing WebRTC

Master's Thesis Defense Presentation

By Dennis McMeekan, School of Computer Sciences

Western Illinois University

Layout

We will be discussing an:

Introduction to the WebRTC specification.

The importance behind this study.

What motivated exploring WebRTC.

The security concerns that will be discussed.

Related research that set the groundwork for this study.

The problem statement that is the center of research and development.

The first security concern which is confidentiality violations.

WebRTC API that allowed for prototype and project implementation.

The idea of covert channels and establishing this with WebRTC.

The research method that set the basis for testing and analysis.

CONTINUE ON NEXT PAGE

An unsecure and secure prototype implementation introducing covert channels and delay mechanisms.

Performance results from the research method and prototype implementation.

The second security concern which is IP Leaks.

A mitigation method to IP Leaks, a Distributed Hash Table server.

The conclusion of this study and possible future work.

And finally, the references which were used.

Introduction

WebRTC is a Real-Time Media Communication Specification

Real-Time Communication allows for two or more individuals to connect in real-time without a delay.

The WebRTC application is a Google product that does not require added plugins or extra installations.

The main development of this specification is surrounded upon making an open-source and royalty-free product that any individual can use to add real-time communication to a web application.

Being royalty-free, Google holds no accountability.

WebRTC is a relatively new specification and not deeply explored.

Due to this, there are security concerns.

Importance

WebRTC is a vital specification due to the increase in growth of media communication. Especially with COVID-19

This can be seen used throughout classrooms, healthcare, businesses, social media, and other areas across the world.

WebRTC is free and available to all developers or individuals.

Motivation

We were motivated to find a new and not fully examined specification.

This allows an establishment of a strong lasting impact on Computer Science and current or future Real-Time communication specifications.

More than ever, there has been a drastic increase in virtual communication and virtual meetings.

WebRTC allows for Security Concerns to be Explored in Relation to Real-Time Communication.

Then we can mitigate these concerns, which will be established later in the presentation.

Security Concerns

We have chosen two specific security concerns due to time constraints:

One being confidentiality violations.

The second being IP leaks.

There are a number of other potential concerns:

Such as cross-site scripting attacking the network connection,

Malware facilitation through data transfer between clients,

Multiple sessions of the same client throughout one connection concurrently.

Infiltrating credential storage of previously logged in clients,

Attacking the network of a WebRTC application.

These security concerns are all important, because of WebRTC's wide availability, users and clients can be exposed to the above security risks.

Related Research

In determing next steps of research, development, and implementation it is vital to understand what has come before this study.

The first development to express is Constructivism in Computer Science Education

This discusses the idea that students are able to use teachers, researchers, and leaders as a source for growth and new horizons. Instead of simply using prior knowledge with no further establishments being made.

The second development to express is A Note on the Confinement Problem

This is a very classic paper, that establishes that there must be a confinement of data within a computer system or program.

CONTINUE ON NEXT PAGE

This was the first research development that brought towards the idea of covert channel implementation.

The last development is a paper that discusses many security vulnerabilities in relation to WebRTC.

This lays out the previous security concerns expressed, and discusses the problem, the outcome, and possible mitigation methods.

Specifically, Covert Channels and IP Leaks which will be both discussed further in the discussion.

Problem Statement

"WebRTC has the foundation to allow for a secure and simple connection to be made by two users without installing native apps or plugins."

Two direct questions in which research and development was based upon,

Is WebRTC susceptible to covert channels?

And how to mitigate this if so,

And is WebRTC susceptible to IP Leaks?

And how to mitigate this if so.

To answer these questions, the issues at hand, confidentiality violations and IP leaks were researched, implemented through prototypes, and discussed.

Confidentiality Violations

With the problem statement being established, we can begin to dive into the security concerns.

Confidentiality Violations relate to a Peer-To-Peer Connection being made where a transfer of real-time data is at the center.

The great benefit of having open-source API is great for development, but can be quite concering considering hackers and exploitations.

This leads to covert channel implementation.

How WebRTC Works

WebRTC is a browser specification.

This means it should be used in partner with web applications.

WebRTC allows for Two or More Clients to Communicate in Real-Time.

This involves a simple web server in which Google has created STUN/TURN protocols that allow for clients to connect via a series of requests and API calls.

When looking at the diagram, we can see two separate clients, a phone, and a laptop, connecting and sending media to each other over a secure network using WebRTC.

WebRTC API

WebRTC API is extremely detailed and resourceful, many webpages detail how to develop web applications of this specification, including a specific API detail provided by Google.

This allowed for research and holding a base understanding.

From this, it is possible to look into real-life examples.

And then from examples, prototype implementation.

Two main languages are the focus, JavaScript and HTML5.

The picture provides a brief overview of the type of API being used on both the client side (left and right) and the server (middle)

Covert Channels

Covert Channels are established in systems or programs in which data can be sent unintentionally.

This can be due to unexpected results of the specfication, in our prototype and thesis this is due to WebRTC API.

This can be harmless, but can also be harmful due to possible sending of Sensitve Data or Malware.

Covert Channels - How They Work

Covert Channels work through exploiting certain elements.

Specfically, WebRTC API to send/receive data

We can look at a simple example,

This example establishes a bit being received based on certain instances of check, create, or delete file.

We take this similar approach, but instead use the bitrate at which data is being sent and received, and then we can receive data based on this.

Covert Channel Implementation

We implement covert channels through a JavaScript Method, Image Filtering. This is commonly used for applying filters on top of a video element, or to possibly alter the data being sent and received.

Instead, we take a similar approach but use a series of delays. This allows for the misuse of this method and function, and allows for bits to be secretively sent and received.

We have two separate connections, local and remote. (representing two clients)

With image filtering, we add an extra step of a canvas element.

The canvas element takes the incoming video element, transfers the data to the canvas element, and then outputs the video from the canvas.

CONTINUE ON NEXT PAGE

Once the canvas element was established, a delay was implemented that allowed for the receiveing end (remote) to sense that delay throughout the flucating bit rate.

If a delay was sensed, a one was received.

If a very small delay was sensed, a zero was received.

Research Method

The center of research was conducted on the idea that the effectiveness of exfiltrating information via covert channels needs to be determined and analyized.

Two main statistics being looked at are the:

Error Rate: How many bits received match the bits being sent

Covert Channel Bandwidth: How much time does it take for each bit or piece of data to be sent and received

It is presumed that an increase in covert channel bandwidth may result in higher covert channel data transfer rates, but an increase in error rate.

It is vital to find the sweet spot, where a reasonable rate of data exchange is found, but also minimal error rates.

CONTINUE ON NEXT PAGE

This led the creation of two separate prototypes:

Unsecure: Covert Channels without Mitigations

Secure: Mitigating Covert Channels

Testing used two Separate “Video” Testing Procedures

One being Black Screen: This is where the webcam is covered and provides consistent data rates, but can have consistent data compression which may provide unrealistic results.

The second being Sample Presentation: This is where the Webcam is not covered, with myself acting like I am giving a presentation. This is a real-life example that helps minimize data compression techniques and provide a realistic scenario.

The rates are averaged because from testing it was determined that the differences are minimal.

For our demonstration, sample presentation will be used to ensure a real-life example is seen.

Unsecure Prototype

The unsecure prototype is centered around implementing covert channels.

This occurs from sending and receiving a bit based on a delay.

Using image filtering, a delay is implemented, then the delay is sensed from the remote or (receiving) client, and the bit is then received.

There are a couple main elements being looked at

- Error rate, covert channel bandwidth, and
- T or the Input Rate: used to determine covert channel bandwidth, and specifcally how often a bit is being sent.

Unsecure Demo

Both demos are conducted using a recorded video because there is an issue that occurs when using my network and home setup while being on a Zoom or Google Meet the bitrates fluctuated to an extreme low or high amount.

We can view the console log, as myself is using the sample presentation method, where a bit is being sent, a delay will occur, and this will affect the bitrate, in which the bit can then be received.

Input Buffer and Output Buffer are just the same as being the Input and Output.

Used the .25 covert channel bandwidth (about 4 seconds) because as we will see later, this is the sweet spot for covert channel implementation.

Results Unsecure Baseline

These were examined without using any covert channel implementation or delays and provided a baseline for judgement.

The Bitrate sits around 1450, and the framerate just slightly above 50.

Results Framerate – Baseline vs Unsecure

The left is the previous framerate seen from the baseline measurements, and the right is the framerate seen with the unsecure prototype implementation.

We do lose some performance, as the framerate does drop about 10-15 frames.

This is minimal in implementing covert channels, as with a real-time communication application this is not that bad.

Results Error Rate – Covert Channels

The error rate is the most important element, because this determines effectiveness of exfiltrating data using covert channels.

We determine that roughly 2-5 seconds is the ideal time to send/receive data, as this held the lowest error rate.

This is because when there is less than 2 or more than 5 seconds, the error rate increases. This is believed to arise from there not being enough time to determine the delay. Or it is too long that the delay actually stabilizes because with the WebRTC specification the delay will slowly begin to correct itself if it is not constant.

Secure Prototype - Mitigation

The secure prototype involves mitigating covert channels with delay mechanisms.

One is using a constant delay – Consistent Delay to provide constant noise.

The second is a random delay – Random Delay to provide random, yet still constant noise.

This simulates control by the admin in mitigating covert channels.

This can possibly result in performance concerns.

But ideally, this will increase the error rate.

Secure Demo

Very similar to the unsecure version, but instead we see added noise between the bits being sent and received.

We are able to see that sometimes the bitrate will remain lower, this is due to the fact that when sending a constant or random delay, there is constant noise in which performance is affected.

But the error rate does increase in terms of going from 0 to 40.

Ideally, this would be more towards 100% as that is perfect results, but for proper examination and to give you all a proper example it is vital to use the same covert channel bandwidth seen with the unsecure prototype.

Results Error Rate – Constant Delay: Secure

The left is the error rate seen previously with the unsecure method, with the right being the secure method implementation.

The error rate does generally increase for each Covert Channel Bandwidth.

But there are certain points that are seen where the error rate from the secure method is a tad bit is less than the unsecure method.

This is believed to be when adding constant noise, the sending and receiving method of covert channel implementation can be fooled into thinking a bit has been received as a one or zero.

This is because we only use two bits, while having a baseline bit to reset and ensure this does not occur would be ideal.

This is a small change but needed to be discussed.

Results Framerate – Constant Delay: Secure

The left is the framerate seen from the unsecure prototype, while the right is the results from the constant delay prototype.

The video quality is a bit worse in comparison to the two prior results, but this is a small tradeoff for implementing mitigation mechanisms.

Results Error Rate – Random Delay: Secure

The left is an examination of the previous error rate seen with a constant delay, while the right is the error rate seen with the random delay.

These result in very similar outcomes, but a random delay mechanism allows for the exploiter to not easily being able to sense this rate.

It was also important to notice performance while consistently implementing a random delay mechanism.

Results Framerate – Random Delay: Secure

This again is a comparison from the previous results of the constant delay, and we can determine that there is a small drop off in performance due to consistently determing a random delay mechanism, but very minimal.

IP Leaks

IP Leaks is a large security concern because the WebRTC specification requires that each client allows for their public IP address to be known.

This is vital in connecting to the other client and using STUN/TURN Protocols.

Because the public IP address can be obtained for harm, this can lead to sensitive data being discovered, such as the geographical location of a client.

WebRTC Current Approach: Server Implementation

As stated in the previous slide, the WebRTC specification uses STUN/TURN protocols to send and receive HTTP requests through a series of a web sockets.

While this ensures reliability and efficiency, this does require the public IP address to be known by each client to connect with the server.

From the image seen, this process can be looked at from a high level.

WebRTC New Approach: Distributed Hash Table

IP Leaks can be mitigated with a Distributed Hash Table.

This type of implementation has been seen with the TOR browser and can replace the previously seen server implementation.

This involves a similar process, but instead of having a central server, a series of relay nodes are used.

DHT Implementation

This graphic provides a high-level example on how this process would work.

Instead of the central server, we can see a series of nine relay nodes.

This can be broken down into three levels with 3 nodes in each: enter, transfer, and exit nodes.

So, we can see two separate clients, Alice, and Jane. And Both will attempt to connect to one of the nodes at each level in sequential order.

This allows for a completely private connection because each relay node does not need to know where the data is coming from, only where to relay the information.

This works extremely well with the TOR browser, but this is commonly seen to download or upload data.

CONTINUE ON NEXT PAGE

Being a real-time communication specification, this could decrease performance to the point where it is not realistic.

But, with this implementation the issue of IP leaks will be resolved.

Proper testing and implementation is needed.

Conclusion

In conclusion, we have determined WebRTC is susceptible to Covert Channels.

This process involves using image filtering through a JavaScript canvas element to allow for fluctuating bitrates involved with data transfer.

Because of this, a bit can be exfiltrated and received between one or more clients.

To mitigate covert channels, two different methods were tested.

Both were delay mechanisms, one being a constant delay, and another being a random delay.

The delay mechanism being random is a better choice in comparison to constant, because being random this will make it extremely more difficult to sense the noise and alter the covert channel accordingly.

CONTINUE ON NEXT PAGE

Ideally, an adaptive method in which the bitrate can be monitored between two clients can be looked at to increase or decrease noise to help mitigate covert channel implementation.

We have also determined that WebRTC is susceptible to IP Leaks.

With this concern, a possible route of mitigation would be to implement a Distributed Hash Table server.

Future Work

An extension of the study to our implementation, would be to use a real network system, where two clients are connected using separate networks and web browsers.

This differs from our approach as we simulate the connection using WebRTC API using one browser and network to aid in testing and implementation.

Our approach still holds validity, because the WebRTC API requires a working network connection, and will still connect through STUN/TURN protocols.

The only real difference being there are not two separate networks.

This may increase or decrease the bitrate seen, in which the covert channel implementation would require a small tweak to how the bit is sensed and received.

CONTINUE ON NEXT PAGE

Another great extension of our work would be to properly implement a prototype of a Distributed Hash Table server for the WebRTC specification.

And with this, full development, testing, and analysis.

Questions:

A **large delay and small delay** are both used because there needs to result in some fluctuation of the bit rate to receive a bit. If only one or the other were used, there would be no direct change to sense the bit on the receiving end.

With low error rates, **error correcting methods** and products exist to solve this problem to make this go even lower.