ENABLE SECURITY

DTLS “ClientHello” Race Conditions in WebRTC Implementations

Alfred Farrugia - Chief Demolition Officer / Senior Researcher

Sandro Gauci - Chief Mischief Officer / CEO

Enable Security GmbH

October 2024

Abstract

This research uncovers a security flaw in WebRTC implementations, urging developers to enforce stricter source verification for DTLS `ClientHello` packets to prevent denial of service attacks.

Introduction to WebRTC

Web Real-Time Communication (WebRTC) is a technology that enables peer-to-peer communication directly between web browsers or mobile applications. It facilitates real-time audio, video, and data sharing without plugins or external software and is considered the most secure open standard VoIP protocol. WebRTC supports secure and efficient media communication by incorporating standardised protocols and encryption methods, such as Datagram Transport Layer Security (DTLS) and Secure Real-time Transport Protocol (SRTP). Media traffic is typically transmitted using the User Datagram Protocol (UDP) to ensure low-latency communication.

In WebRTC, a secure media session is initiated by a sequence of three events: initial signalling, media consent verification, and DTLS handshake. These events are mostly abstracted from developers and are typically managed by browser APIs, following the guidelines set in IETF’s RFC 8829 - JavaScript Session Establishment Protocol (JSEP)[1].

During the initial signalling phase, the peer initiating a WebRTC session takes the first step by sending an offer to another peer, requesting a media session to be established. If the targeted peer accepts the offer, an answer is sent back, marking the beginning of the communication. This offer-and-answer exchange is typically carried out over HTTPS or WebSocket connections, using standard protocols (such as SIP) or other proprietary protocols. Several important parameters are exchanged during the signalling phase, which inform the subsequent two phases: ICE media consent verification and DTLS handshake. For the problem explained in this paper, the following parameters are relevant for the media consent verification phase: ICE candidates, ICE username fragment and ICE password. ICE candidates, containing an IP address and port, are used to discover the possible network paths between WebRTC peers. The ICE username fragment and the ICE password authenticate messages used during media consent verification. For the DTLS handshake, the following parameters are essential: DTLS role and certificate fingerprint. The DTLS role is either “active” or “passive”; where the “active” peer acts as a Client and sends the initial `ClientHello` while the “passive” peer acts as a Server and expects this initial `ClientHello`. The certificate fingerprint will be used later in the DTLS handshake to verify the validity of the client’s certificate and to create a cryptographic binding between the exchanged information and the SDP.

Media consent verification uses the ICE candidate information to verify that both WebRTC peers can (a) communicate with each other via a reliable path and (b) that the other party is the peer they claim

[1] https://datatracker.ietf.org/doc/html/rfc8829

to be. The Interactive Connectivity Establishment protocol (ICE) uses Session Traversal Utilities for NAT (STUN) messages, which contain a message integrity attribute calculated by the ICE username fragments and ICE password obtained in the signalling phase. This communication is typically carried out over UDP using an ephemeral port.

Once this verification is complete, a DTLS handshake is performed over the network path established during media consent verification. The DTLS handshake begins with the peer having the “active” role, sending a `ClientHello` message with the proposed security parameters. The passive peer responds with a `ServerHello` message, agreeing on parameters, followed by its certificate and key exchange. The active peer verifies the passive peer’s identity and sends its key exchange message so both parties can generate the session keys. Finally, both exchange “Finished” messages, confirming the encryption setup and completing the secure connection. When the peers send their respective certificates, the other peer validates that certificate against the certificate fingerprint sent during signalling. The keying material generated by the DTLS handshake is then used for SRTP key derivation.

The following sequence diagram shows the flow of packets for setting up a secure media session between two peers.

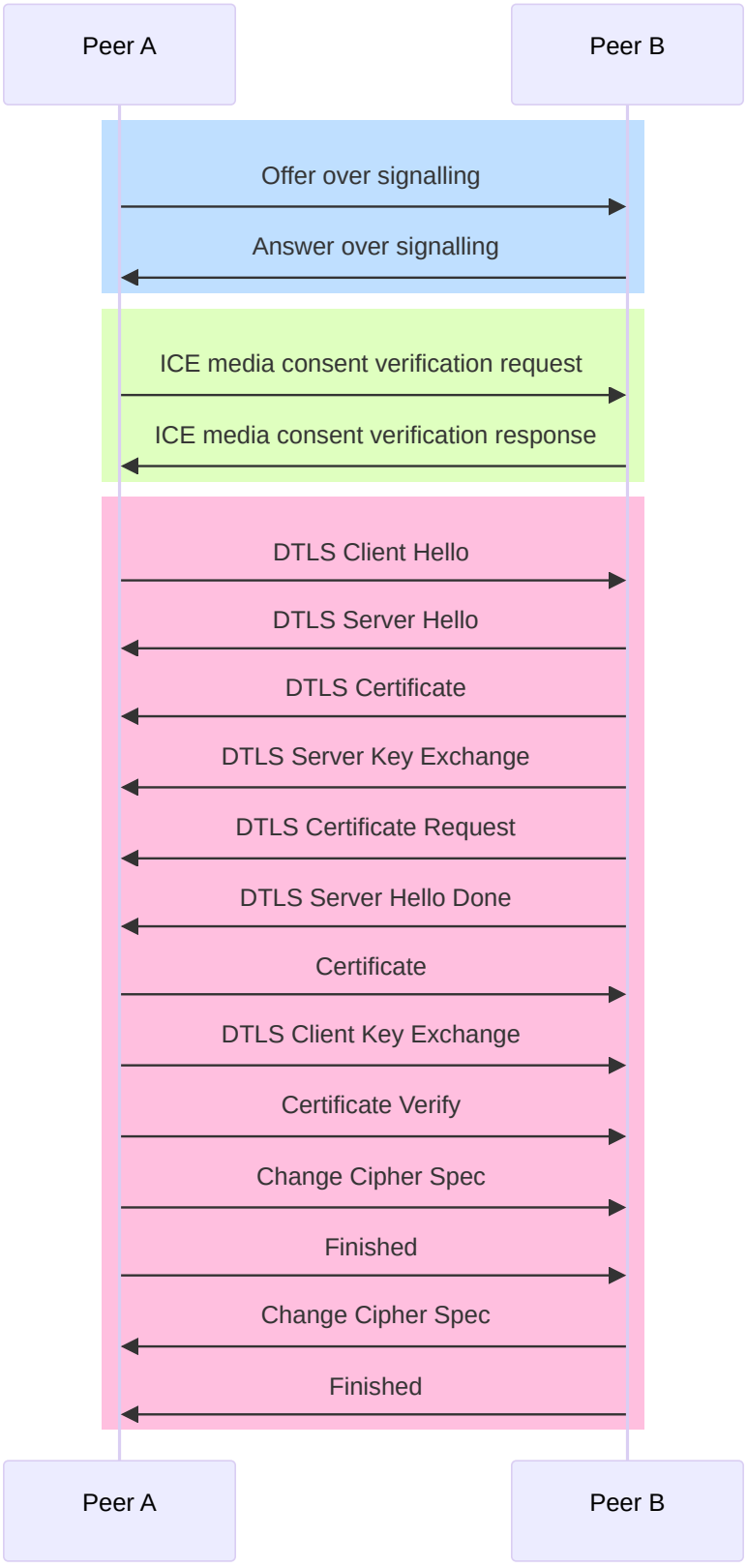

Figure 1: Flow of packets for setting up a secure media session between two peers

Peer A, the initiator, typically acts as a DTLS client, also known as an active DTLS role, by sending `ClientHello`, while Peer B, the responder, take a DTLS server role, also known as a passive DTLS role. Note that the role selection is unrelated to who sends the initial offer over signalling but is based on values exchanged during signalling.

Although WebRTC allows for direct peer-to-peer media connectivity between browsers, intermediary signalling and media servers often handle traffic between peers. There are several reasons why WebRTC services do this, including performance, network adaptability, recording, and NAT traversal. Intermediary media servers handle network communication by either using an ephemeral port for each distinct media session or using one port to handle all the media sessions for all the peers.

The following sequence diagram shows the flow of packets when intermediary signalling and media servers are used:

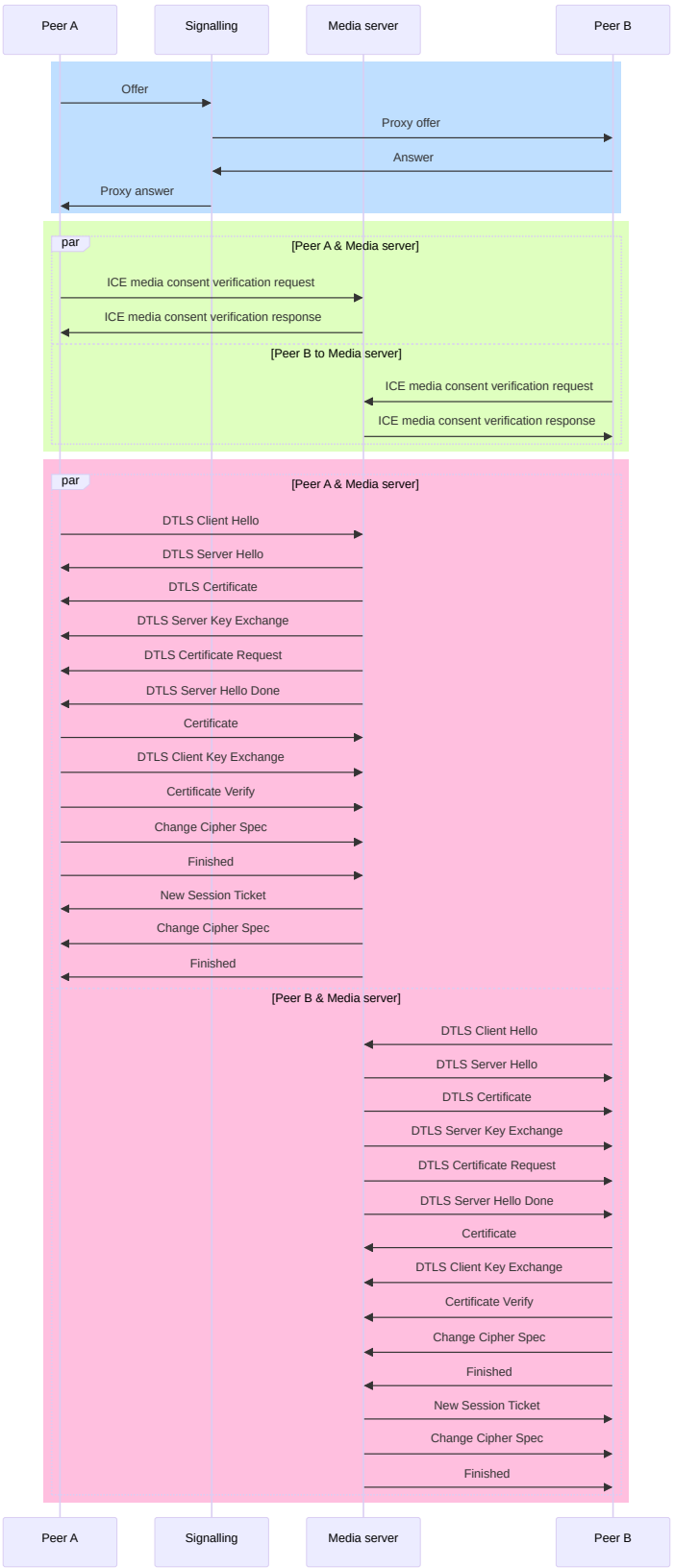

Figure 2: Flow of packets when intermediary signalling and media servers are used

The impact of Denial of Service attacks on WebRTC

Denial of Service (DoS) attacks can significantly impact the reliability and performance of WebRTC applications, leading to delayed or failed connections, unestablished media streams, or a poor user experience. WebRTC systems can be affected by various types of DoS attacks, including volumetric attacks that overwhelm network capacity and targeted attacks that, while not volumetric, exploit specific vulnerabilities to cause service disruptions. The issue described in this paper falls into the latter category.

The issue explained

Our research has identified a gap between the media consent verification and the DTLS handshake phase. The communication takes place over UDP, which does not inherently verify the authenticity of the packet’s source unless additional checks are implemented at the application layer. This means that an adversary can send a malicious DTLS `ClientHello` message from any IP address to the expected port, potentially causing a “network race condition” if the malicious message is processed before the legitimate one. Several implementations were found to act this way, assuming the packet’s source is legitimate. This often results in a Denial of Service, especially when the malicious `ClientHello` message contains a list of insecure cipher-suites like `TLS_NULL_WITH_NULL_NULL`.

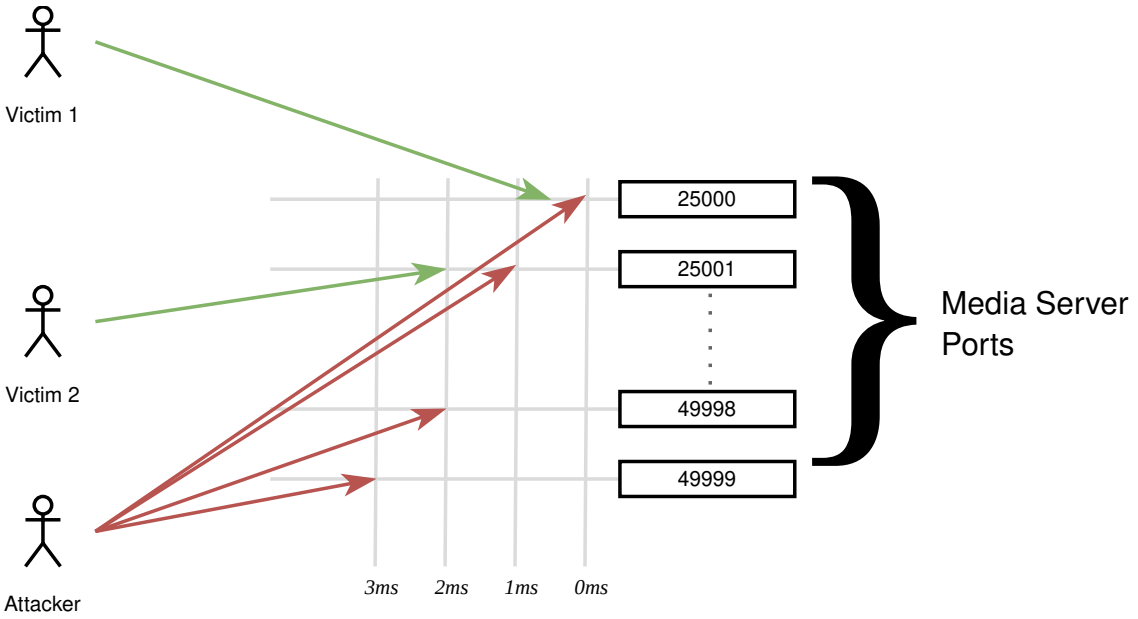

Figure 3: This diagram illustrates a network race condition scenario between legitimate and malicious packets arriving at a media server. The red arrows represent the attacker’s packets, while the green arrows represent legitimate traffic from victims. The x-axis quantifies time in milliseconds (ms), showing when packets reach the server ports. The media server ports range from 25000 to 49999. The attacker’s packets arrive earlier (closer to 0ms on the x-axis), indicating that they reach the server before the legitimate traffic, which arrives later at 0.5ms. This timing discrepancy highlights a race condition where the attacker’s packets potentially exploit vulnerabilities by arriving first, disrupting or taking priority over the legitimate traffic from the victims. The diagram effectively shows how the attacker can gain an advantage by manipulating packet arrival times.

The study observed different behaviours when testing various open-source and proprietary WebRTC solutions. In some cases, a message sent over signalling terminates the WebRTC session. In other instances, a DTLS Alert is sent to the peer, ending the media stream without necessarily updating the session state over signalling. This might lead to an undefined application state where the peer incorrectly believes the call is ongoing while the media communication has failed.

This issue does not affect media servers or peers that use a static port since, by design, these systems will check the source of the packets. Interestingly, our research found issues in several media servers but not in Web Browsers.

Special case: when no ICE candidates are used

It is worth noting that in VoIP media sessions, which are not WebRTC media sessions, there may be no ICE candidates during the signalling phase. In such cases, media consent verification is skipped. This configuration can make the media server vulnerable to receiving malicious DTLS `ClientHello` messages on the port expecting legitimate traffic. This behavior has been observed in solutions where media consent verification is not mandatory and the media session relies on DTLS-SRTP.

Related work

The RFC 8826 - “Security Considerations for WebRTC”, section 4.2 - “Communications Consent Verification” highlights the importance of the ICE handshake, referring to presumption of malicious traffic:

> It is important to remember here that the site initiating ICE is presumed malicious; in order for the handshake to be secure, the receiving element MUST demonstrate receipt/knowledge of some value not available to the site (thus preventing the site from forging responses).

RFC 8827 - “WebRTC Security Architecture”, section 4.2, also describes the use of ICE for media consent verification, which ensures that both peers are willing and able to exchange media. Section 4.3 discusses the DTLS handshake, highlighting the sequence dependency between ICE completion and the DTLS handshake.

However, these RFCs do not specifically address the check on the source of the DTLS `ClientHello`’s message.

Methodology

To test our hypothesis, we tested against open-source and proprietary WebRTC implementations in the form of media servers. The initial tests were conducted against open-source projects using SIP as their signalling protocol. We implemented two simple Golang programs, one acting as the peer (using SIP protocol) while the other acts as an attacker. These two tools interacted with each other to simulate an attacker’s `ClientHello` being processed before the legitimate peer’s `ClientHello`. The target media server under test was set to auto-answer incoming offers to facilitate testing. The following steps were performed:

1. The SIP client sent an offer over WebSocket to initiate a media session, including ICE parameters, DTLS role (set to active) and DTLS Certificate fingerprint
2. The target answered the offer
3. Media consent verification was performed

4. The SIP client sent a message to the attack tool with the details of where the DTLS `ClientHello` was expected
5. The attack tool started to send a DTLS `ClientHello` with the list of cipher-suites set to `TLS_NULL_WITH_NULL_NULL` to the target every 10ms
6. The SIP client sent a DTLS `ClientHello` message to the target, including a valid list of cipher-suites

These steps reproduced the vulnerability described in this paper, in a controlled manner. Another test was performed by continuously sending a DTLS `ClientHello` against a port range on the target media server. This test proved that the issue can be abused in the wild and not just in a controlled lab environment.

However, these tools were not sufficient for testing proprietary and cloud implementations. These solutions use custom and complex signalling protocols, and conducting tests that could cause a denial of service on production systems is risky. To test these systems safely and efficiently, we modified the Chromium browser to notify the attack tool with the remote peer’s ICE candidates just before it sends its DTLS `ClientHello` message. Specifically, we patched the `JsepTransport::AddRemoteCandidates` function to send a POST request to the attack tool. The attack tool would then begin to send the malicious DTLS `ClientHello` before the browser does, targeting only the media port assigned to the media session under test. This way, other users of the system under test were not affected.

Case Studies

In this section, we present case studies of various software and platforms to highlight different behaviors and scenarios. Each case study focuses on a specific software or platform to illustrate the diverse behaviors and scenarios that can occur in systems found to be vulnerable to this security issue.

WebRTC Media Sessions with ICE where signalling and a DTLS Alert message terminate the Media Session

This behavior was observed when testing Asterisk, an open-source software PBX used to manage and control telephone calls between traditional phone sets, PSTN destinations, and VoIP network devices or services. The tested versions of Asterisk were 18.20.0, 20.5.0 and 21.0.0, all vulnerable to this issue. Signalling between the peer and the Asterisk server was performed over secure WebSocket using the SIP protocol. Media consent verification was performed as soon as the peer received the server’s answer. This was followed by a `ClientHello` message sent from a malicious actor, i.e., an IP and port different from the one used in the media consent verification phase. The call was immediately termi-

nated: the peer received a SIP BYE message from the Asterisk server, with the Reason header set to “`Q.850;cause=0`”, and a DTLS Alert message terminating the DTLS handshake.

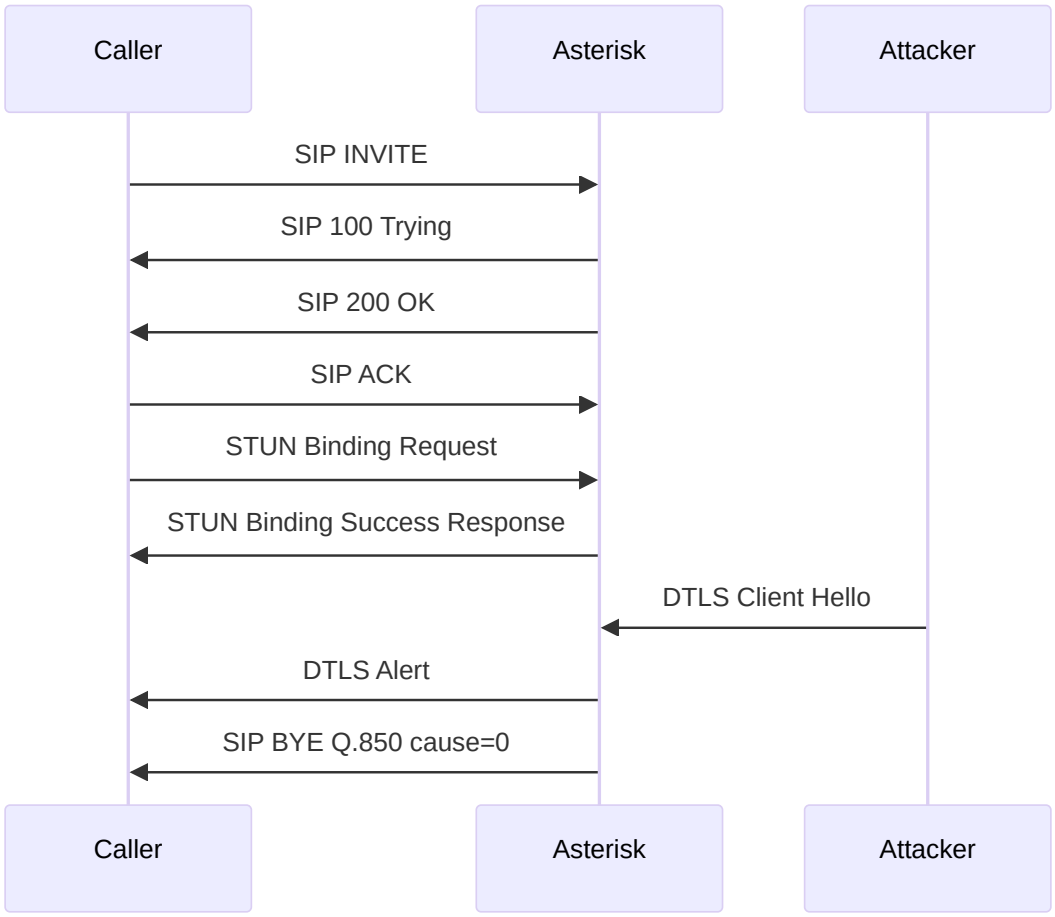

Figure 4: WebRTC Media Sessions with ICE where signalling and a DTLS Alert message terminate the Media Session

WebRTC Media Sessions with ICE where neither signalling nor a DTLS alert message terminates the Media Session

This behaviour was observed when testing RTPEngine, a proxy for RTP traffic and other UDP based media traffic. The tested version of RTPEngine was mr11.5.1.6, which was vulnerable to this issue. For testing, the signalling was handled by a Kamailio server. Media consent verification was performed as soon as the peer received the server’s answer. This was followed by a `ClientHello` message sent from a malicious actor, i.e., an IP and port different from the one used in the media consent verification phase. An error was immediately reported in the RTPEngine logs. However, the peer received no DTLS

alert or signalling messages, leaving the call in an undefined state.

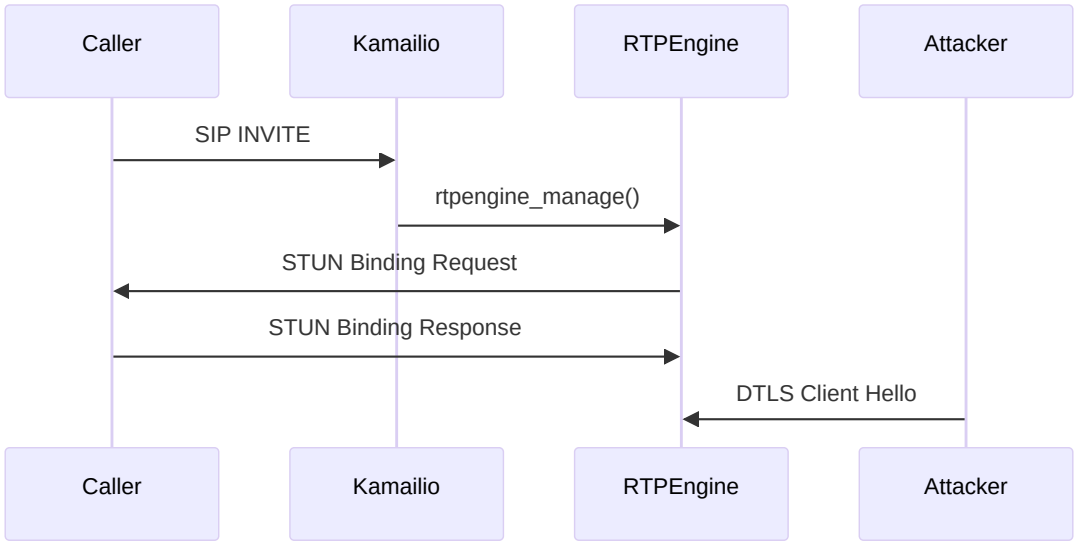

Figure 5: WebRTC Media Sessions with ICE where neither signalling nor a DTLS alert message terminates the Media Session

The following log shows that RTPEngine resets the DTLS connection context:

```
DEBUG: [... port 39910]: [srtp] Processing incoming DTLS packet
ERR: [... port 39910]: [crypto] DTLS error: 1 (no shared cipher)
ERR: [... port 39910]: [srtp] DTLS error on local port 39910
DEBUG: [... port 39910]: [crypto] Resetting DTLS connection context
```

VoIP Media Sessions without ICE for DTLS-SRTP encrypted calls

This behaviour was observed when testing FreeSWITCH, an open-source telephony software for real-time communication protocols using audio, video, text and other forms of media. It was found to be vulnerable to this issue, even when no ICE candidates were present. The tested version of FreeSWITCH was 1.10.10. Signalling between the peer and the FreeSWITCH server was performed over secure WebSocket using the SIP protocol. When the peer received the server’s answer, a `ClientHello` message was sent from a malicious actor, i.e., an IP and port different from the one specified in the signalling phase. The call was immediately terminated: the peer received a SIP BYE message from the FreeSWITCH server, with the Reason header set to “`Q.850;cause=27`”, and a DTLS Alert message terminating the DTLS handshake.

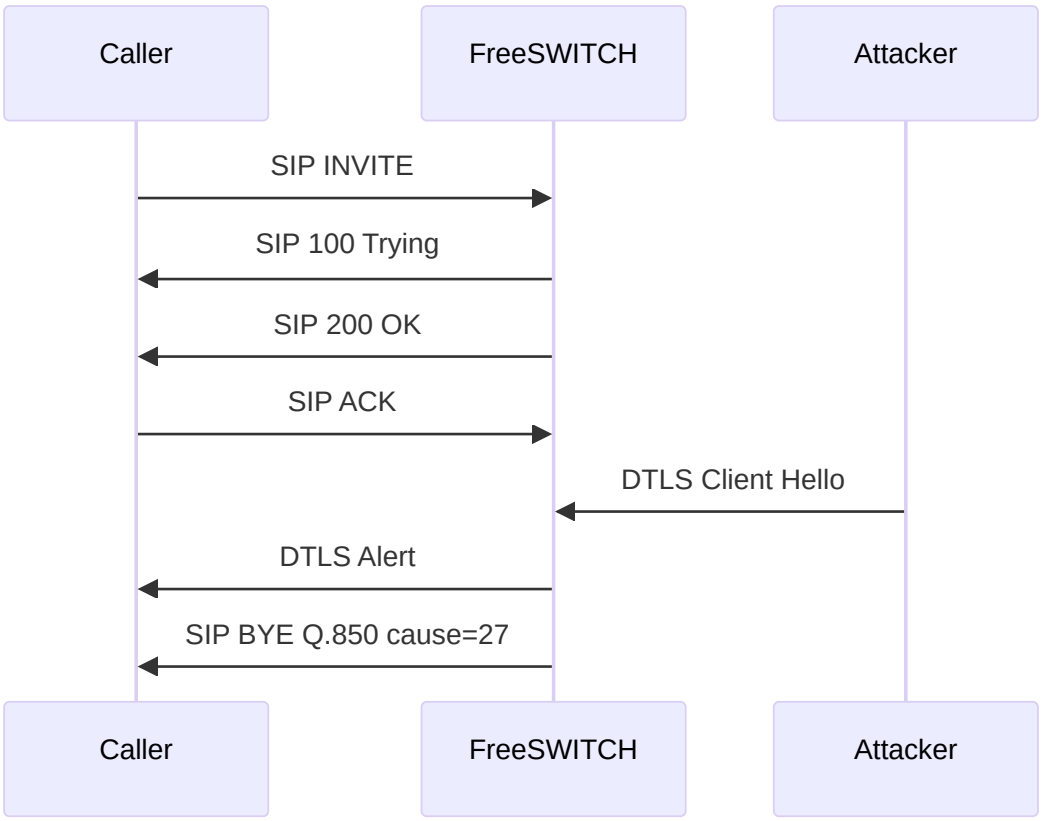

Figure 6: VoIP Media Sessions without ICE for DTLS-SRTP encrypted calls

WebRTC Media Sessions with ICE when making PSTN Calls

This behavior was observed when testing Skype, a telecommunications application operated by Skype Technologies, a division of Microsoft. While the platform was generally resilient against this attack during calls between Skype users, vulnerabilities were identified under specific circumstances. Reproducing this issue between two Skype users using the web client proved to be challenging, as peer-to-peer communication was typically chosen as the nominated candidate pair, which was not susceptible to the attack.

However, the vulnerability was successfully exploited when users engaged in calls from Skype to PSTN or when using the “Skype Test Call” service (echo123). The media servers handling these calls were found to be vulnerable to this issue as of July 2024. Specifically, the vulnerability was confirmed in two scenarios:

1. When the victim user placed a call to a mobile number (PSTN).
2. When the victim user initiated a call to the “Skype Test Call” service (echo123).

During the attack, the calls failed to establish correctly, with neither DTLS Alert messages nor any visual indications of an error appearing in the user interface.

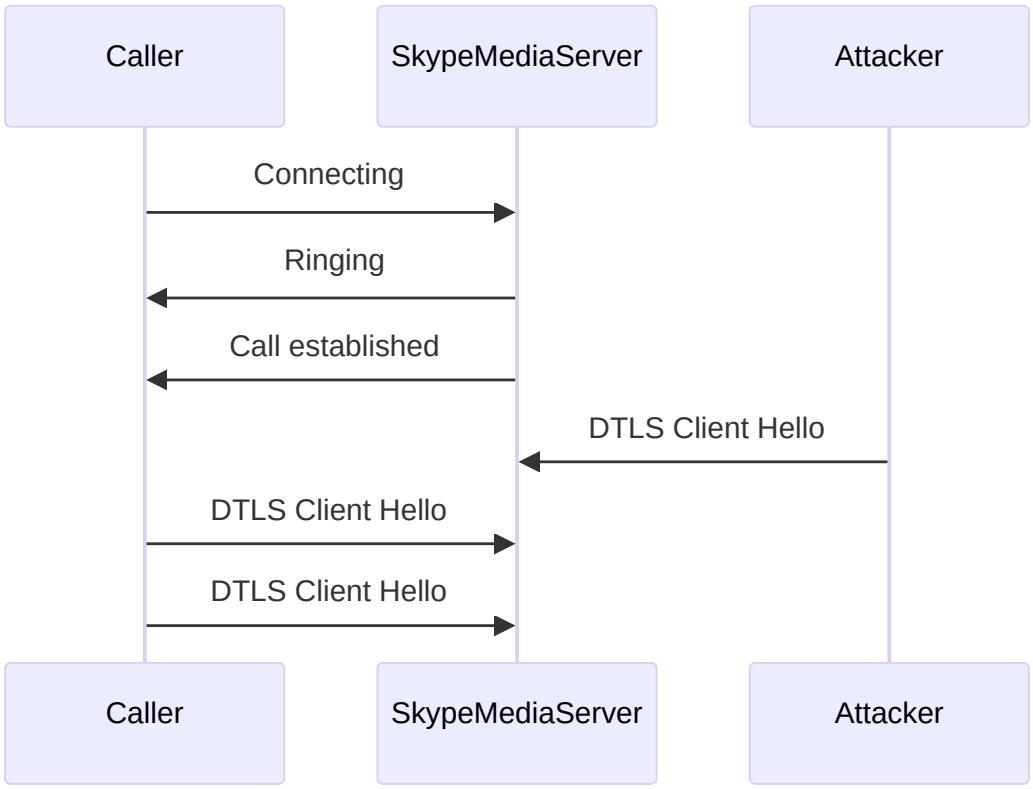

Figure 7: WebRTC Media Sessions with ICE when making PSTN Calls

In both scenarios, the vulnerability did not produce explicit alerts or error messages for the user, leaving the issue undetected during the active call session.

Results

The following list shows several open-source and proprietary solutions affected by this issue and their behaviour during attack.

Product	Signalling Disconnects	Receives DTLS Alert
Asterisk (<=18.20.0, <=20.5.0, <=21.0.0)	✓	✓
FreeSWITCH (<=1.10.10)	✓	✓

Product	Signalling Disconnects	Receives DTLS Alert
RTPEngine (<mr12.1.1.2, <mr12.0.1.3, <mr11.5.1.16, <mr10.5.6.3, <mr10.5.6.2, <mr9.5.8.2, <mr8.5.12.2)	×	×
Skype	×	×

During testing, we also examined solutions that were not susceptible to this vulnerability. Many of these solutions treated ICE (Interactive Connectivity Establishment) as a transport mechanism. The table below outlines various factors implemented by both open-source and proprietary solutions to prevent this vulnerability. These measures were observed in systems that were found to be secure at the time of testing.

Product	Peer is in active mode	Non-ephemeral port
Janus	✓	×
Discord Service Voice channel	✓	×
Dolby.io Live Broadcast	✓	×
Facebook Messanger web client	✓	×
Google Meet	✓	✓
LiveKit Meet [2]	×	✓
Webex Meetings	✓	✓
Zoho Meeting	×	×
Zoom personal room meeting	✓	✓
Mediasoup	✓	✓

Discussion

The behaviour discussed in this paper does not imply that a bug exists in the WebRTC specification. However, it highlights a gap that multiple implementors have failed to identify: checking the origin of the DTLS `ClientHello`. Since these protocols are used in real-time communication services, disruption in media sessions leading to denial of service should be considered critical.

[2] only DataChannel is set to active

When ICE candidates are present, the solution to this problem should be simple: ensure that the source of the DTLS `ClientHello` packet is the same as the one used during the media consent verification phase. However, solutions might be harder to implement when no ICE candidates are present. One potential solution is to match the source of the DTLS `ClientHello` packet with the information exchanged during signalling.

During our research, it became evident that although RFC 8826 and RFC 8827 refer to performing checks on the media traffic, it is not clear that the DTLS `ClientHello` message should be processed only if its source matches the selected ICE candidate pair. We recommend that future RFC updates include a specific check for the source of the DTLS `ClientHello`.

Through discussions with key figures in the WebRTC community[3], it became evident that a core issue behind the vulnerability described in this paper appears related to a narrow interpretation of “media” in WebRTC contexts. A common assumption, particularly among those with a VoIP background, is that media refers exclusively to RTP. RFC 8445, section 12.2, “Receiving Data,” focuses on RTP and RTCP verification, which can lead to the mistaken belief that other protocols, such as DTLS and SCTP, are not subject to ICE verification. This ambiguity around the handling of non-RTP media can result in the assumption that ICE only governs RTP and RTCP. The absence of explicit guidance for DTLS and other protocols in media contexts has contributed to the view that DTLS falls outside the scope of ICE-verified media.

Conclusion

The findings of this study underscore a security vulnerability in current WebRTC implementations, specifically the gap between media consent verification and the DTLS handshake phase. This vulnerability, primarily arising from UDP’s lack of inherent packet source verification, poses a significant risk of disruption and denial of service in real-time communication services. Specifically, it is to be found when a WebRTC implementation treats ICE *only* as an initial consent mechanism, whereas when a WebRTC implementation threats ICE as a transport mechanism, this security issue is mitigated by design.

To mitigate this vulnerability, developers must implement stricter checks on the source of DTLS `ClientHello` packets, ensuring they match the verified ICE candidate. This additional layer of verification is crucial in preventing malicious actors from exploiting this gap. Alternative verification mechanisms based on signalling information should be developed and integrated for scenarios where ICE candidates are not present.

The observed inconsistencies in behaviour among different WebRTC solutions highlight the need for

[3] https://www.enablesecurity.com/newsletter/2024-06-rtcsec-news/#heated-debate-on-whether-the-webrtc-specs-contain-a-vulnerability

a unified approach to address this issue. Industry-wide collaboration and adherence to updated standards can facilitate the adoption of these essential security measures. Including explicit guidelines in future RFC 8826 and RFC 8827 updates will provide clear directives for developers, ensuring comprehensive protection across all implementations.

Thanks

The authors would like to express their sincere gratitude to Iñaki Baz Castillo, Richard Fuchs, Philipp Hancke, Dan Jenkins, Olle E (oej) Johansson, Tsahi Levent-Levi, Lorenzo Miniero, Nils Ohlmeier, Tim Panton, Torrey Searle, Dr. Ing. Dorgham Sisalem and Markus Toepfer for their invaluable contributions to this white paper. Their expertise, insights, and dedication were instrumental in shaping the content and ensuring its accuracy and relevance. We are deeply appreciative of their time, effort, and unwavering support throughout the development of this white paper.

About Enable Security

Enable Security is a dedicated team of security researchers specializing in Real-Time Communication (RTC) security. We are driven by the belief that communication is a fundamental human right, and by securing it, we empower people to communicate freely without fear or constraint.

Our passion for RTC security stems from two core principles: the importance of protected communication in today’s digital world and our love for tackling complex challenges. We approach our work with a combination of expertise, curiosity, and a commitment to continuous learning.

At Enable Security, we pride ourselves on:

1. Delivering high-quality, valuable results through meticulous research and analysis
2. Fostering a culture of constant learning and knowledge sharing within the security community
3. Maintaining honesty and transparency in our assessments and communications
4. Being approachable and collaborative, working seamlessly with colleagues, clients, and the broader security ecosystem

Our team is composed of friendly, passionate professionals who are always eager to engage with new ideas and challenges. We believe in the power of teamwork and open dialogue to drive innovation and improve security practices across the RTC landscape.

By choosing Enable Security, you’re partnering with a group of dedicated experts who are committed to advancing the field of RTC security and protecting the fundamental human need for safe, secure communication.

RTCSec Newsletter

Stay informed with the RTCSec newsletter, a free periodic publication that delivers insightful commentary and news on VoIP and WebRTC security. We cover both defensive and offensive security aspects related to Real-time Communications. Subscribe now to stay ahead in the field: RTCSec Newsletter Subscription[4]. You can also read it online: RTCSec Newsletter Online[5].

References

Alvestrand, Harald T. “RFC 8825: Overview: Real-Time Protocols for Browser-Based Applications.” *IETF Datatracker*, 2021, datatracker.ietf.org/doc/html/rfc8825. Accessed 6 Sept. 2024.

Ari Keränen, et al. “RFC 8445: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal.” *IETF Datatracker*, 2018, datatracker.ietf.org/doc/html/rfc8445. Accessed 6 Sept. 2024.

Enable Security. “Asterisk Susceptible to Denial of Service via DTLS Hello Packets during Call Initiation.” *GitHub*, 15 Dec. 2023, github.com/EnableSecurity/advisories/tree/master/ES2023-01-asterisk-dtls-hello-race. Accessed 6 Sept. 2024.

—. “FreeSWITCH Susceptible to Denial of Service via DTLS Hello Packets during Call Initiation.” *GitHub*, 22 Dec. 2023, github.com/EnableSecurity/advisories/tree/master/ES2023-02-freeswitch-dtls-hello-race. Accessed 6 Sept. 2024.

—. “RTPEngine Susceptible to Denial of Service via DTLS Hello Packets during Call Initiation.” *GitHub*, 15 Dec. 2023, github.com/EnableSecurity/advisories/tree/master/ES2023-03-rtpengine-dtls-hello-race. Accessed 6 Sept. 2024.

McGrew, David, and Eric Rescorla. “RFC 5764: Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-Time Transport Protocol (SRTP).” *IETF Datatracker*, 2024, datatracker.ietf.org/doc/html/rfc5764. Accessed 6 Sept. 2024.

Mozilla. “WebRTC API.” *MDN Web Docs*, 4 Oct. 2019, developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

Rescorla, Eric. “RFC 8826: Security Considerations for WebRTC.” *IETF Datatracker*, 2021, datatracker.ietf.org/doc/html/rfc8826. Accessed 6 Sept. 2024.

—. “RFC 8827: WebRTC Security Architecture.” *IETF Datatracker*, 2021, datatracker.ietf.org/doc/html/rfc8827. Accessed 6 Sept. 2024.

[4] https://www.enablesecurity.com/subscribe/
[5] https://www.enablesecurity.com/newsletter/