# RTCWEB Security

IETF 83.5

Eric Rescorla

ekr@rtfm.com

# Overview

- Communications consent and DoS prevention
  - Consent freshness–what is required?
  - Simulated forking
  - Traffic rate limiting
- Identity Protocol Details
- Resolution of issues raised in reviews (Thomson, Druta)

# Communications Consent Overview

- Consensus to use ICE for initial consent
  - Sufficient for prevention of cross-protocol attack
  - Not so great protection against packet-based DoS
- Open issues
  - What is required for continuing consent?
  - Should we limit sender rate?
  - What about simulated forking?

# Consent Freshness

- As specified, ICE only checks at start of session
- Keepalives just keep the NAT binding open
  - But aren't confirmed
  - Or authenticated
- What if I no longer wish to receive traffic?
  - General agreement, some sort of keepalive
  - Check every $X$ seconds
  - If I don't receive a keepalive after $Y$ seconds must stop transmitting
    * Can re-start ICE if needed

# Concrete Proposal:

## `draft-muthu-behave-consent-freshness-00`

- Defines a new ICE method, `Consent`
  - Simple request/response
  - No username/password or message integrity
- Why not just use STUN binding Request?
  - Binding requests require integrity checks

  "One of the reasons for ICE choosing STUN Binding indications for keepalives is because Binding indication allows integrity to be disabled, allowing for better performance. This is useful for large- scale endpoints, such as PSTN gateways and SBCs as described in Appendix B section B.10 of the ICE specification."

# draft-mutha Consent Algorithm

1: **repeat**
2: $T \leftarrow now()$
3: $Failures \leftarrow 0$
4: **repeat**
5: $T2 \leftarrow now()$
6: Start STUN Consent transaction
7: **if** Success **then**
8: go to 16
9: **end if**
10: $Failures \leftarrow Failures + 1$
11: **if** $Failures == 3$ or $T_M < (now() - T)$ **then**
12: Stop transmitting; exit
13: **end if**
14: Wait until $T2 + T_c$
15: **until** False
16: Wait till $T + T_c$
17: **until** Call ends.

- Proposed values: $T_m = 30, T_c = 15$

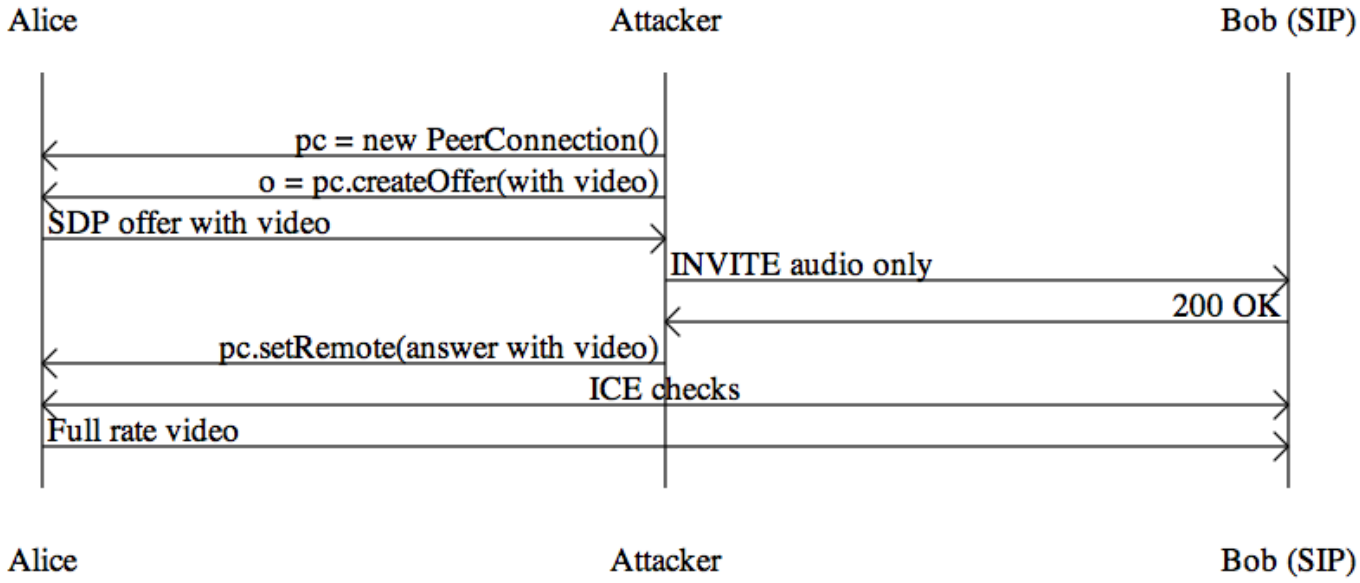
# Duration of Unwanted Traffic

- Assume we have initial consent and then receiver goes offline
  - On average next check will be at $T_c/2$ (worst case $T_c$)
  - Takes around $T_m$ to fail
    - Complicated interaction of $T_c$, $T_m$ and ICE RTO
    - Worst-case is about $2T_m$
- With RFC 5389 parameters, a transaction fails after $39500$ ms
  - Expected duration of unwanted traffic is $47$ s
  - This seems awful long
- We probably to shorten these parameters

# Is a MAC needed?

- ICE Binding Requests are authenticated with a MAC
  - Based on ufrag and password
- `Consent` as currently specified does not have a MAC
  - All security is from the 96-bit STUN transaction ID
  - ... must be pseudorandomly generated
  - This is plenty of security against an off-path attacker
- An on-path attacker can simulate consent even if the victim is not responding
  - MAC requires attacker to have username and password as well
- Not clear if there is a concrete attack that requires MAC

# DoS via Excessive Traffic



# Why does this work?

- ICE only verifies connectivity
  - But anything can be sent over that channel
- SDP parameters are under the control of the attacker
  - And that is what controls bit rate

# No user consent required (?)

- We just need a source of high bandwidth data
  - We (probably) can't use a datachannel because it's congestion controlled
  - And the sequence numbers are unpredictable (allegedly)
- But media probably is not
- It's not video that requires user consent
  - ... but access to the camera
- Set up a bogus MediaStream blob that generates continuous patterns
  - Use it to source the data
  - This shouldn't trigger consent dialogs

# How serious is this issue?

- Basically another version of voice hammer
- Short duration
  - If we have consent freshness then $< 1$ minute
- But very scalable
  - I can mount this using an ad network or any other traffic fishing system
  - No user consent required
  - Not self-throttling (unlike HTTP-based attacks)

# ... But

- ICE implementations already need to implement Binding Requests

  "Though Binding Indications are used for keepalives, an agent MUST be prepared to receive a connectivity check as well. If a connectivity check is received, a response is generated as discussed in [RFC5389], but there is no impact on ICE processing otherwise." [RFC 5245; Section 10]

- So Binding Requests will work better with non-RTCWEB equipment

# A related problem: liveness testing

- Applications want to detect connection failure
  - This needs to happen on a shorter time scale than consent
  - How much dead air will people tolerate? ($< 5seconds$)
- Proposal: configurable minimal *received* traffic spacing
  - If no packet is received in that time, send a Binding Request
  - Application failure signaled on Binding Request failure

# Combined Consent/Liveness Proposal I: Shorten STUN timers

- Proposal: $Rc = 5$, $Rm = 4$. Use measured RTO, minimum $200ms$
- Example: Packets transmitted at $0, 200, 600, 1400, 3000$; transaction fails at $4600ms$
- Rationale
  - If our RTT is $> 5s$, that's not going to be a very good user experience

# Combined Consent/Liveness Proposal II: Both checks done via binding requests

- Consent timer: $T_c$ (default $= 20s$, no more than $30s$)
- Packet receipt timer: $T_r$ (no less than $500ms$); configurable
- When either timer expires start a STUN transaction
- When a STUN transaction succeeds, re-start both timers
- When a STUN transaction fails
  - If transaction was started by $T_c$, stop sending, abort
  - ... else, notify application of failure, but keep sending
- $T_r$ is also reset by receiving any packet from the other side

# What API points do we need?

- Ability to set keepalive frequency (individually on each stream?)
- A consent transaction has failed and so I am not transmitting on stream $M$
- A liveness check has failed on stream $M$

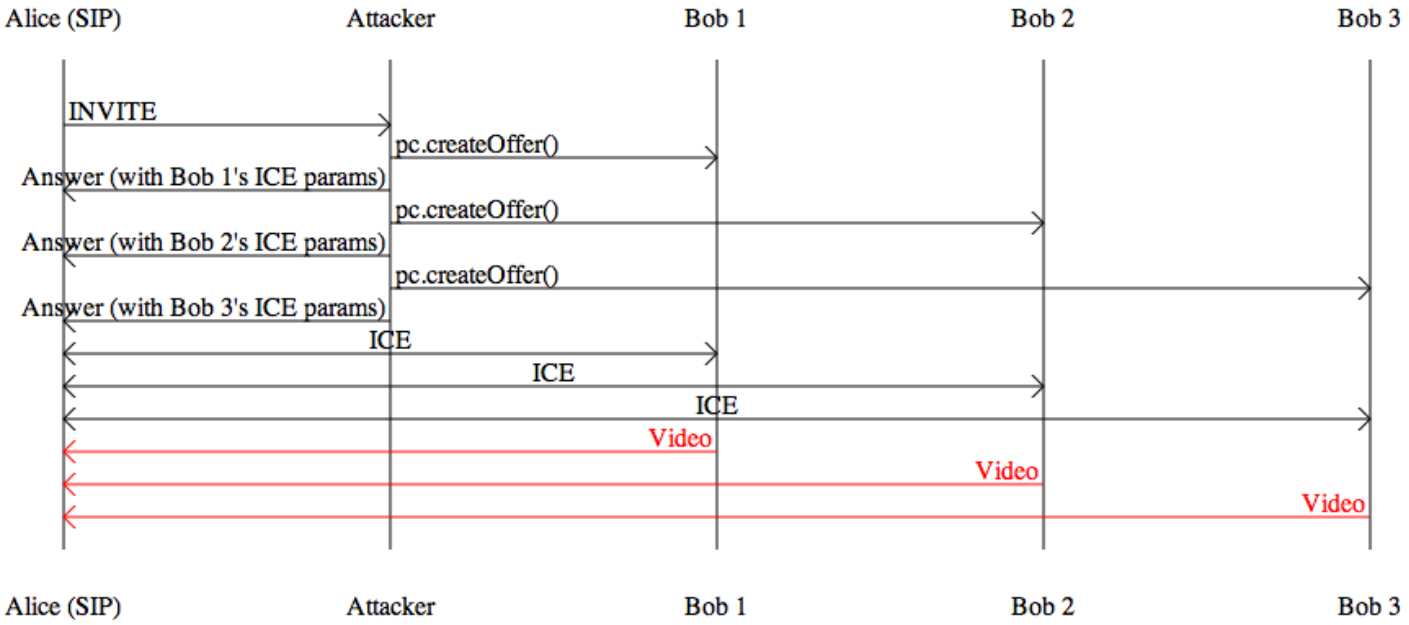
# Proposed API

```
// Do a liveness check every 500ms and call callback if it fails
pc.setLivenessCheck(500, m, function(m) {
   // media stream m has apparently failed
});


//
pc.onstreamfailed = function(m) {
   // media steam consent check has failed
}
```

- Would a constraint + event combination be better?

# Simulated Forking [Westerlund]



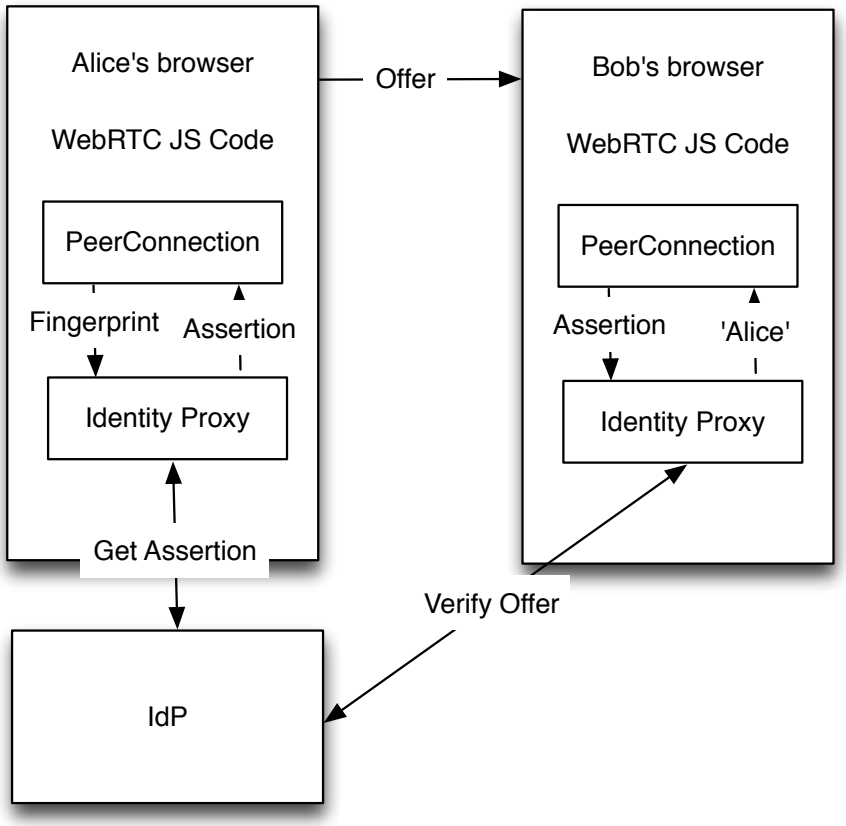
# Proposed Solutions

- Rate limit the number of outstanding ICE connections you respond to?
  - Based on unique ufrag/passwords
- If using DTLS you can rate limit the number of DTLS associations
  - Not clear that this is better than ICE

# Reminder What are we trying to accomplish with Identity?

- Allow Alice and Bob to have a secure call
  - Authenticated with their identity providers
  - On any site
    - Even untrusted/partially trusted ones
- Advantages
  - Use one identity on any calling site
  - Security against active attack by calling site
  - Support for federated cases

# Reminder: Overall Topology



# What needs to be defined

- Information from the signaling message that is authenticated [IETF]
  - Minimally: DTLS-SRTP fingerprint
  - Generic carrier for identity assertion
  - Depends on signaling protocol
- Interface from PeerConnection to the IdP [IETF]
  - A specific set of messages to exchange
  - Sent via `postMessage()` or WebIntents
- JavaScript calling interfaces to PeerConnection [W3C]
  - Specify the IdP
  - Interrogate the connection identity information

# Data to be authenticated

- JSON dictionary with one value: fingerprint

```
{
   "fingerprint":
     {
       "algorithm":"SHA-1",
       "digest":"4A:AD:B9:B1:3F:...:E5:7C:AB"
     }
}
```

- Note: this format is trivially the same as the a-line format

# Wire format for identity assertions

```
"identity":{
     "idp":{     // Standardized
        "domain":"idp.example.org", // Identity domain
        "method":"default"          // Domain-specific method
     },
     "assertion": "..."             // IdP-specific
}
```

# Open issue: how is identity assertion carried?

- Delivered separately to application
  - Requires application to manage the data
- Along with SDP in `createOffer()/createAnswer()`
  - This only works with the "dictionary" form
  - And doesn't guarantee fate-sharing with SDP
- Best option: put it in an a-line
  - Fate-shares with SDP
  - Can tag to individual a-lines if necessary
  - Potentially SIP compatible (though not with existing endpoints)

# Concrete Proposal: Opaque Value

- Just base-64 the data and shove it in an a-line
  - e.g., "identity"
  - Like ICE candidates can apply to entire SDP or individual m-lines

```
a = identity: <base-64ed of JSON>
```

- Alternative: actually render the identity information
  - But we need to potentially escape stuff anyway, so encapsulated is easier

# A concrete example

```
v=0
o=- 1181923068 1181923196 IN IP4 ua1.example.com
s=example1
c=IN IP4 ua1.example.com
a=setup:actpass
a=fingerprint: SHA-1 \
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=identity: 6XCJib2JAZXhhbXBsZS5vcmdcIiwKICAgICAgICAgICAgICAg\
            ICAgICAgICAgXCJjb250ZW50c1wiOlwiYWJjZGVmZ2hpamtsb\
            W5vcHFyc3R1dnd5
t=0 0
m=audio 6056 RTP/AVP 0
a=sendrecv
a=tcap:1 UDP/TLS/RTP/SAVP RTP/AVP
a=pcfg:1 t=1
```

# Generic Downward Interface (Implemented by PeerConnection)

- Instantiate "IdP Proxy" with JS from IdP
  - Probably invisible IFRAME
  - Maybe a WebIntent

    `https://<idp-domain>/.well-known/idp-proxy/<protocol>`
- Send (standardized) messages to IdP proxy via `postMessage()`
  - "`SIGN`" to get assertion
  - "`VERIFY`" to verify assertion
- IdP proxy responds
  - "`SUCCESS`" with answer
  - "`ERROR`" with error

# Signature Messages

```
PeerConnection -> IdP proxy:
  {
    "type":"SIGN",
     "id":1,
     "contents":
       "{\"fingerprint\":{\"algorithm\":\"SHA-1\",
       \"digest\":\"4A:AD:B9:B1:3F:...:E5:7C:AB\"}}"
  }

IdPProxy -> PeerConnection:
  {
    "type":"SUCCESS",
    "id":1,
    "message": {
      "idp":{
        "domain": "example.org"
        "protocol": "bogus"
      },
      "assertion": "..." // opaque
    }
  }
```

# Verification Process

```
PeerConnection -> IdP Proxy:
  {
    "type":"VERIFY",
    "id":2,
    "message": "..." // opaque
  }

IdP Proxy -> PeerConnection:
  {
   "type":"SUCCESS",
   "id":2,
   "message": {
     "identity" : {
       "name" : "bob@example.org",
       "displayname" : "Bob"
     },
     "contents":
       "{\"fingerprint\":{\"algorithm\":\"SHA-1\",
       \"digest\":\"4A:AD:B9:B1:3F:...:E5:7C:AB\"}}"
     }
  }
```

# Meaning of Successful Verification

- IdP has verified assertion
  - Identity is given in "identity"
  - "name" is the actual identity (RFC822 format) [OPEN ISSUE]
  - "displayname" is a human-readable string
- Contents is the original message the AP passed in

# Processing Successful Verifications

- Authoritative IdPs
  - RHS of `identity.name` matches IdP domain
  - No more checks needed
- Third-party IdPs
  - RHS of `identity.name` does not match IdP domain
  - IdP MUST be trusted by policy
- These checks performed by PeerConnection

# New API points needed

- Set the desired IdP and identity
- Pre-generate an assertion (performance)
- Interrogate a validated assertion

# Set desired IdP and identity

```
void SetIdentityProvider (
                           DOMString Provider,
                           optional DOMString protocol,
                           optional DOMSTring username
                          );
```

`provider` – domain name of the provider

`protocol` – the protocol name (locally meaningful)

`username` – the desired user name

- OPEN ISSUE: relationship to user settings (who wins)?

# Pre-Generate Assertions

```
void GenerateIdentityAssertion (
                                  AssertionSuccessCallback success,
                                  PeerConnectionErrorCallback error
                                );
```

- Idea here is to generate identity assertion ahead of time
  - Same reason as we want to start ICE gathering early
- This works because we only need DTLS fingerprint at this time
- What should be passed to the callback? anything?

# Interrogate a valid assertion

```
readonly attribute IdentityAssertion peeridentity;

interface IdentityAssertion {
  attribute DOMString name; // The peer identity (rfc822-style)
  attribute DOMString displayname; // Human-readable
};
```

- If identity was not used, this attribute is null
- OPEN ISSUE: Do we need an event for identity verification?

# Recap of reviews

- Reviews from Martin Thomson, Dan Druta
- Latest drafts attempt to address their issues
- I probably missed stuff

# HTTPS versus HTTP Threat Model

> "Obviously, the standard class of problems with unsecured HTTP exist, but within the context of this application, there aren't that many more that this enables. The example in S4.1.3 is not unique to this application. It applies to any user consent that is tied to a particular web origin.
>
> In comparison to possibly visiting and _using_ a site operated by a web attacker, this is not substantially worse, or requiring significantly more effort to analyze.
>
> Of course, the only safe assumption is that you are talking to a web attacker when using unsecured HTTP." – Thomson

New text:

> Conventionally, we refer to either WEB ATTACKERS, who are able to induce you to visit their sites but do not control the network, and NETWORK ATTACKERS, who are able to control your network. Network attackers correspond to the [RFC3552] "Internet Threat Model". Note that for HTTP traffic, a network attacker is also a Web attacker, since it can inject traffic as if it were any non-HTTPS Web site. Thus, when analyzing HTTP connections, we must assume that traffic is going to the attacker.

# Consent

> "I think that it has been well-established that consent is required for access to input devices (e.g., camera/microphone). The implication from S4.1 is that this is sufficient as well as necessary. There is one crucial piece of the argument that is absent:
>
> > A site with access to camera or microphone could send media either to itself or any site that indicates consent (see CORS). Sending media over HTTP or thewebsocketprotocol is likely to perform less well than is ideal, but it would work." – Thomson

Some new text here:

> It's important to understand that consent to access local devices is largely orthogonal to consent to transmit various kinds of data over the network (see Section 4.2. Consent for device access is largely a matter of protecting the user's privacy from malicious sites. By contrast, consent to send network traffic is about preventing the user's browser from being used to attack its local network. Thus, we need to ensure communications consent even if the site is not able to access the camera and microphone at all (hence WebSockets's consent mechanism) and similarly we need to be concerned with the site accessing the user's camera and microphone even if the data is to be sent back to the site via conventional HTTP-based network mechanisms such as HTTP POST.

# Calls from non-same-Origin IFRAMEs

"I've seen a range of responses from sites. Some sites vet advertisers carefully, others could care less as long as the money flows. Those that vet have usually been stung once already.

If reputation is important to you, then it is your responsibility to safeguard your own reputation. If you rely on others, then you can use technical measures (checking, etc...), or simply rely on their own desire to safeguard their reputation.

In summary, I don't see a need for a specific technical solution to this problem."

- Should non-same-origin IFRAMEs notify the enclosing frame
- Proposed resolution: remove the text

# IP Address Hiding (relay)

- Current requirements
  - API to suppress negotiation until call is answered (can be done with SDP editing)
  - API to suppress non-relay candidates (now done via constraints)

    "Section 5.4 stipulates requirements that I don't think are reasonable. Including discussion on the subject, including countermeasures, with a conclusion that there are no requirements on the API would be good. However, guidance for site implementers would be sensible." – Thomson
- Note sure how to proceed here

# Persistent Consent and HTTP

"You already established that - in the presence of a network attacker - consent to foo.com is equivalent to consent to bar.net. Therefore, it seems reasonable to regard the two as being equivalent. Once you make that leap, then it is easy to see that you don't have enough granularity to make any consent meaningful. So you have to conclude that providing a (persistent) consent for non-HTTPS sites is pointless." – Thomson

- I don't agree they are equivalent
  - Web attackers do exist
- Do we want to forbid persistent consent for HTTP?

# Identity and Linkability

> "What about identity? While you are in the business of creating an identity system, wouldn't it be nice if the site you are using couldn't identify the people using that site? Imagine that you are able to create an assertion that you are dhouhqed08gslkn209eejit8sfsdo@rtfm.com (that well known IdP), which is translated (by the IdP validator) to a ekr@rtfm.com only in the browser chrome..."

- This is compatible with the proposed identity system
- Though requires that the IdP validator verify the RP's identity
- I'll add some text

# Access to Local Devices

"Since this describes the threat I think is important to point out the fact that even when the user provides consent it is difficult for them to determine the real reason they give the consent to a site. Take the scenario in which a site obtains consent from the user for an app that captures a clip and saves it on the user's hard drive. Later on, if the user gave permanent consent to the site, the site can obtain access to the camera for the purpose of streaming without the user knowing that. This can be confusing for users even if they trust the site." – Druta

- General limitation of technical consent mechanisms
  - Once I let you take my media and send it somewhere it's hard to constrain what you do with it
- Same thing is generally true with, e.g., image upload, Facebook wall
- This is what privacy policies are for