Proposed WebRTC Security Architecture

IETF 82

Eric Rescorla

ekr@rtfm.com
Trust Model

- Browser acts as the *Trusted Computing Base* (TCB)
  - Only piece of the system user can really trust
  - Job is to enforce user’s desired security policies

- Authenticated entities
  - Identity is checked by the browser (sometimes transitively)

- Unauthenticated entities
  - Random other network elements who send and receive traffic
Authenticated Entities

- Examples:
  - Calling services (known origin)
  - Identity providers
  - Other users (when cryptographically verified)
  - Sometimes network elements with the right topology (e.g., behind our firewall)

- Authenticated ≠ trusted: Dr. Evil is still evil even if I know it’s him
  - But authentication is the basis of trust decisions
  - And maybe I want to call Dr. Evil after all...
Unauthenticated Entities

- Pretty much anyone else
  - Generally cannot be trusted
- But can still be used when behavior can be verified
  - ICE reachability testing
  - Transit data which is cryptographically verified
Basic Design Principle: As good a job as we can

• It’s always safe to browse the Web
  – Even to malicious sites

• Calls are encrypted wherever possible
  – At minimum between WebRTC clients unless the site takes direct action [Open issue warning]

• When available directly verify the far side
  – Minimizes required trust in calling site
  – Be compatible with as many identity providers as possible
Overall Topology

IETF 82
WebRTC Security Architecture 6
Call Flow (I)

- Bob knows Alice is calling [verified with IdP]
  - Browser can display trusted UI for Alice’s identity
  - If in address book, maybe name, picture, etc.
- If no IdP, Bob knows signaling service claims Alice is calling
Call Flow (II)

- Alice knows Bob has answered
  - Verified with Bob’s identity provider
- Alice and Bob know media is not flowing to innocent third parties (media consent)
- Alice and Bob know they have a secure call with each other
  - Security details displayed via trusted UI
Permissions Models

- One-time camera/microphone access [MUST]
- Permanent camera/microphone access (scoped to origin) [MUST]
- User-based permissions [SHOULD]
  - Allow calls to this verified user
  - Allow calls to any verified user in my system address book (on some set of sites?)

- Data channels MAY be created without user consent
Permissions API

• MUST provide a mechanism to distinguish permissions type
  – E.g.,
    new PeerConnection({permission:’PERMANENT’, ...})
  – Allows the browser to display different UIs for each permissions level

• MUST provide a mechanism to relinquish any media stream access
  – E.g., via MediaStream.record()
  – Allows a site to commit not to observing your data
  – Needs to be reflected in a trusted UI
Who “owns” the permissions”

• Question: which operation triggers the permissions check?
  - mediaStream creation
  - peerConnection.addStream()
  - peerConnection.setLocalDescription()
  - peerConnection.setRemoteDescription()

• This has UI and programmer implications

• An even bigger issue if API doesn’t work in terms of SDP at all
Permissions UI

- MUST clearly indicate when the camera/microphone are in use
- SHOULD stop camera and microphone when UI indicator would be masked
  - E.g., window overlap
- SHOULD provide a distinctive UI when user’s identities are directly verifiable
Why HTTP origins are a problem

• Assumption: I’ve authorized http://www.example.com

• I’m in an Internet Cafe and visit any URL
  – Attacker injects IFRAME pretending to be PokerWeb
  – But calls go to him

www.slashdot.org

pokerweb.example.org

new PeerConnection() {
  ...
};

• Result: attacker has bugged your computer

• Violates the Web security model
Web Security Issues

- MUST treat HTTP and HTTPS origins as different permissions domains
  - e.g., http://example.com/ and https://example.com/ are different
- Active mixed content MUST NOT be treated as if it were the HTTPS origin
  - [OPEN ISSUE]: How do we do this exactly?
Web Security and State Machine in JS

- Proposal is to split up state machine logic
  - ICE in browser
  - SDP/Media negotiation in JS
  - Develop a library to assist in SDP/Media negotiation

- Where to JS libraries come from?
  - Standard procedure is to download from a CDN
  - E.g.,
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.0/jquery.min.js">
  - At minimum you want HTTPS (not all CDNs do this)
  - CDN is now inside security boundary

- Not clear how different this is
  - Lots of sites use JQuery, underscore, etc. anyway
Communications Consent

- All direct communications MUST be verified via ICE
- The ICE stack MUST be constructed so that the JS cannot obtain the transaction id
  - This means that at minimum STUN must in browser
- Implementations MUST verify continuing consent at least every 30 (?) seconds

**OPEN ISSUE**: How to verify continuing consent?
- ICE keepalives are STUN Binding Indications (one-way)
- Proposal: use STUN Binding Requests instead
IP Location Privacy

- Setting up a direct connection leaks an agent’s IP address
  - And hence information about its location
- API MUST allow suppression of ICE negotiation until the user accepts session
- API MUST provide a mechanism to do TURN-only candidates
  - SHOULD allow conversion to non-TURN once peer identity is verified [Jesup]
- No need to have browser enforce user consent
  - A malicious site can get your IP address anyway
  - If you are running Tor, you want the browser to do media through Tor, though
Communications Security: Implementation Requirements (Proposed)

- MUST implement DTLS-SRTP (for media) and DTLS (for data)
- MAY implement RTP(?) and SDES(??) for backward compatibility purposes
- Security MUST be default state
  - Implementations MUST offer DTLS and/or DTLS-SRTP for every channel
  - MUST accept DTLS and/or DTLS-SRTP whenever offered *

*Somewhat harder with a low-level API, but still possible with the right design.
Communications Security: API Requirements

- Implementations MUST support PFS modes
- Implementations MUST allow JS to force new long-term key generation
  - E.g.,
  ```javascript
  new PeerConnection({new_authentication_key:true,...})
  ```
  - This allows unlinkability
- Implementations SHOULD allow JS to set authentication key lifetime
  - This allows key continuity
- When DTLS is used, API MUST NOT provide access to the traffic keying material
Communications Security: UI [based on draft-kaufman-rtcweb-security-ui]

- MUST provide a security inspector interface in browser chrome
- Up-front items
  - Security characteristics of incoming stream
  - Security characteristics of outgoing A/V
  - Whether the transmission keys were pairwise derived or provided by a server
  - Verified far endpoint identity if available
- With drill-down
  - Cipher suites
  - PFS yes or no
  - Out-of-band verification mechanism such as fingerprint or SAS
Example IdP Interaction: BrowserId

Alice’s Browser

WebRTC JS Code

Peer Connection

BrowserID Signer

Get Certificate

Identity Provider

Bob’s Browser

WebRTC JS Code

Peer Connection

BrowserID Verifier

Signed Fingerprint

'Allice'

Offer

Check Certificate

Signed Fingerprint

Fingerprint
Example ROAP OFFER with BrowserID

{
    "messageType": "OFFER",
    "callerSessionId": "13456789ABCDEF",
    "seq": 1
    "sdp": 
    v=0
    ...
    "identity": {
        "identityType": "browserid",
        "assertion": {
            "digest": "<hash of fingerprint and session IDs>",
            "audience": "[TBD]",
            "valid-until": 1308859352261,
        }, // signed using user’s key
        "certificate": {
            "email": "rescorla@gmail.com",
            "public-key": "<ekrs-public-key>",
            "valid-until": 1308860561861,
        } // certificate is signed by gmail.com
    }
}
Example JSEP Transport Info with BrowserID

{
  "name": "audio",
  "fingerprint": {
    "algorithm": "SHA-1",
  },
  "identity": {
    "identityType": "browserid",
    "assertion": {
      "digest": "<hash of fingerprint>",
      "audience": "[TBD]"
    }, // signed using user's key
    "certificate": {
      "email": "rescorla@gmail.com",
      "public-key": "<ekrs-public-key>",
      "valid-until": 1308859352261,
    } // certificate is signed by gmail.com
  },
  "candidates": [...]
}
Generic Third-Party Identity Assertions [Warning: hard-hat area]

• We don’t want to be tied to any identity provider or protocol
• Best case scenario: accomodate BrowserID, OAuth, OpenID, etc.
  – Without changing browser code
• Basic idea
  – Generic fixed downward interface from PeerConnection
  – IdPs provide adaptation layers to their own protocols
  – Potential avenues:
    ∗ Load JS from a defined place on the site
    ∗ Web intents
• Still working on this part (lots of help from Mozilla guys)
Questions?