MediaStream security

Randell Jesup, Mozilla

IETF/W3C Interim
Jan 2012

Who can access MediaStreams?

- Threat model is that WebRTC apps are not trusted be default
  - “Trusted” or “installed” apps may be given more trust
- If the app is not trusted with security keys, etc we shouldn't trust it with access to the media
  - There are a number of ways an app could mis-use this access
  - If we can't establish a secure connection, particularly DTLS-SRTP with identity, does this add any extra threats?
- How does this affect the MediaStream Processing proposal? http://www.w3.org/TR/streamproc/
- How do we deal with this?

Example of the problem

Using pokerstars example:

Pokerstars may be legit and not snoop (or they may), but their site (and app) may have been compromised by someone who wants to listen in on you and your partner plan strategy in a sidebar while waiting to play.

Money would be at stake, perhaps even large amounts of money == incentive.

Even if the pokerstars app is untrusted and we use DTLS-SRTP with BrowserID, if the app has access to the raw media it could mirror it (perhaps compressed, or after filtering for keywords) to pokerstars, and perhaps in some cases elsewhere (modulo normal cross-origin protections).

There are examples involving use by political resistance movements as well, where the app may well be untrusted.

Trusted & untrusted apps

- “Trusted” apps (via whatever mechanism is chosen to assert trust) could probably be given direct access to the media.
  - Should this be a separate level or permission?
- Untrusted apps would be blocked from doing a lot of transform-like actions on MediaStreams
  - Note that MediaStreams are not just for WebRTC
  - Justin's face-detection demo is an example
  - “Photobooth” app which wants to provide effects/captions
  - This implies the ability to grant access to MediaStreams without access to bypass the user permission for getUserMedia(), either one-time or remembered

Cross-origin protection

- We can block the app from accessing the MediaStreams by leveraging cross-origin protections
- This can be used for blocking indirect access by pulling the data out of a <canvas>, etc
- It can also be used to block direct access to MediaStreams by JS workers such as those in the MediaStream Processing API, effectively sandboxing them
  - They could be allowed to process the stream, but be blocked by cross-origin from communicating this back to the JS app
  - Timing and other attacks via a worker **would** be able to leak data, but the bandwidth of a leak would be extremely limited
  - Predefined processing nodes would still be allowed

Open issues

- What level of MediaStream security is needed?
  - Trusted vs untrusted apps
  - Single level of trust vs separate permissions
  - Does this only make a difference in DTLS-SRTP with identity? (Which is our “secure, private” mode)
  - Does not having this make security/privacy promises unworkable?
- How does this affect MediaStream Processing?
  - Is the amount of information leakage acceptable?
- UI impact and user understanding of the permission model

Discussion