RTC-Web Security Considerations

RTC-Web Interim (June 2011)

Eric Rescorla

ekr@rtfm.com

The Browser Threat Model

Core Web Security Guarantee: "users can safely visit arbitrary web sites and execute scripts provided by those sites." [HCB+10]

- This includes sites which are hosting malicious scripts!
- Basic Web security technique is isolation/sandboxing
  - Protect your computer from malicious scripts
  - Protect content from site A from content hosted at site B
  - Protect site A from content hosted at site B
- In this case we're primarily concerned with JavaScript running in the browser

The browser acts as a trusted computing base for the site

Threat Model

Web Attacker: Operates a malicious Web site.

- Can convince you to go there
- Cannot impersonate some other site.

Network Attacker: Controls your network

- Conventional Internet threat model
- Defended against with cryptographic protocols
  - Unfortunately not universally deployed

Background: The Same Origin Policy (SOP)

- A page's security properties are determined by its *origin* [Bar10b]
  - This includes: protocol (HTTP or HTTPS), host, and port
  - All these must match for two pages to be from the same origin
- Each origin is associated with its own security context
  - Scripts in origin A have only very limited access to resources in origin B
- *Important:* the origin is associated with the page, *not* where the script came from
  - Scripts loaded via `<script src="">` tags are associated with the origin of the page, not the URL for the script!

Background: Same Origin Policy for Page Data

- Scripts can only access page data from their own origin
  - Contents of the DOM
  - JavaScript variables
  - Cookies
  - Important exception: JavaScript pointer leakage [BWS09]
- Scripts can access any other page data from their origin
  - Includes other windows and IFRAMEs
- Frame can navigate their own children
  - This is used for cross-site communication (e.g., FaceBook Connect)

Background: Same Origin Policy for HTTP Requests

- JavaScript can be used to make fairly controllable HTTP requests with `XMLHttpRequest()` API
  - But only to the same origin
- Origin A can make partly controllable requests to origin B via HTML forms
  - But cannot read the response
  - *Cross-Site Request Forgery* (CSRF) defenses depend on this
- Origin A can read scripts from origin B
  - But they run in A's context
  - This is done all the time (e.g., Google analytics)

Browser Security Invariants

- Don't add features that allow the browser to mount new attacks
  - Even against poorly secured systems
- Avoid in-flow "click here to screw yourself" dialogs [Bar10a]
  - Users routinely click through these [SEA+09]
- Default to secure operation
  - Users don't check security indicators [SDOF07]

List of Issues to Consider for RTC-Web

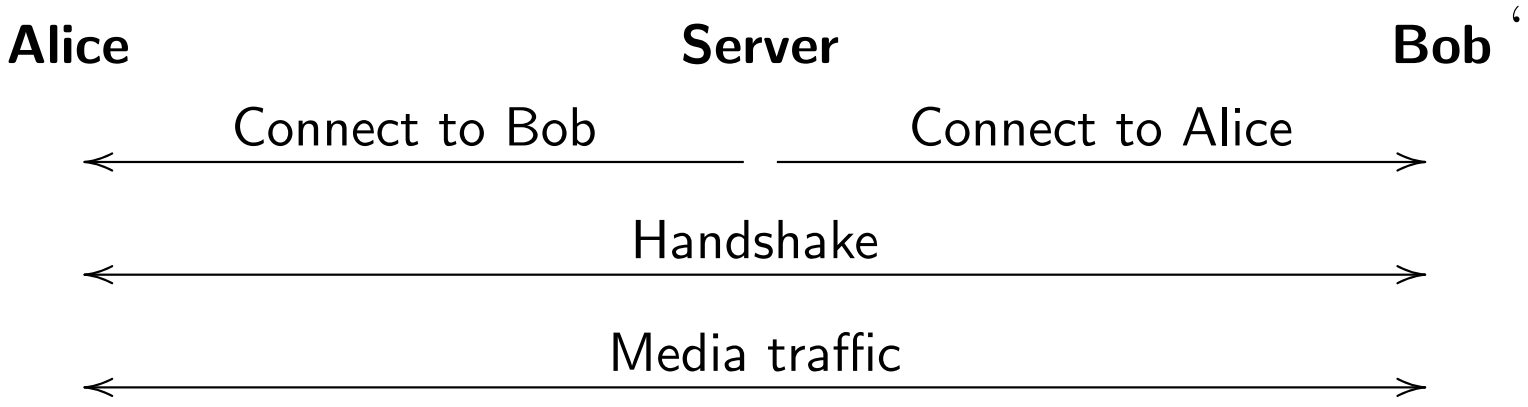
- Consent to communications
- Access to local devices
- Communications security

Consent for real-time peer-to-peer communication

- Need to able to send data between two browsers
  - Unless you want to relay everything
- But this is unsafe (and violates SOP)
  - Not OK to let browsers send TCP and UDP to arbitrary locations
- General principle: verify consent
  - Before sending traffic from a script to recipient, verify recipient wants to receive it from the sender
  - Familiar paradigm from CORS [vK10] and WebSockets[Fet11]

How to verify communications consent for RTC-Web

- Can't trust the server (see above)
  - Needs to be enforced by the browser
- Browser does a handshake with target peer to verify connectivity



- This should look familiar from ICE [Ros10]

Implementing Communications Consent Securely

- Remember: we don't trust the JS

- Restrict pre-handshake communications
  - Restrict communications to an endpoint until handshake completes
  - Minimize application control of ICE packets (extensions, etc.)
  - Rate-limit ICE checks
- Browser **must not** let application see STUN secret
  - Prevents forgery of STUN responses by the server
- What about cross-protocol attacks?
  - Not really an issue for UDP
  - TCP **must** use masking

Access to Local Devices

- Making phone (and video) calls requires that your voice be transmitted to other side
  - But the *other side* is controlled by some site you visit
  - What if you visit `http://bugmyphone.example.com`?
  - All this takes is a web attacker!
- Somehow we need to get the user's consent
  - But to what?
  - And when?
- Approval **must** be scoped to site origin [Bar10b, JB08]

How to get user approval (not totally an IETF issue)

- Remember: need to avoid in-flow dialogs
  - Consent cannot be obtained for each call
- Most likely need to a get approval ahead of time
  - E.g., via an application "install" experience for each site
- Browsers **should** have clear indicators that you are in a call
  - **Should not** be maskable by web application
  - E.g., part of browser chrome
  - But remember users mostly won't check
- Once a site is approved you need to mostly trust it

Local Device Access and Network Attackers

- Say I have approved device access for `http://www.example.com/`
  - I visit `http://www.example.com/` over an insecure network
  - Attacker injects his own code and initiates a call to himself
  - This attack can persist even after I change networks ("origin infection")
- Sites **should** offer RTC-Web only over HTTPS
  - HTTP and HTTPS are different origins
- Browsers **should** forbid RTC-Web access in mixed content settings
  - ... when consent is for HTTPS but some JS is fetched via HTTP

What about communications security?

- **Must** provide security against message recovery and message modification
  - For both media (voice/video) and data
  - All the usual protocols work fine for this part
- What about threats by the calling service itself?
  - Controls nearly all the UI
  - Direct interaction with the browser difficult [Bar10a]
- Potential attacks by the calling service

  Retrospective: The calling service is is non-malicious during a call but is subsequently compromised (preventable)

  During-call: The calling service is compromised during the call it wishes to attack (hard to prevent)

Protecting Against Retrospective Attack

- Assume attacker has access to encrypted media stream
- If calling service has access to traffic keys, attack is trivial
  - Even worse in Web contexts because of extensive logging
  - Hard to believe service can adequately "forget" keys it has seen
    - Most sites log requests at many different locations
- Right approach: public key-based exchange between the endpoints
  - Secure against retrospective attack even if mediated by calling service
  - APIs **must not** allow calling service to subsequently extract traffic keys
  - Best if it provides perfect forward secrecy (PFS)

Protecting Against During-Call Attack

- Need to have a public key exchange
  - Otherwise passive attack is trivial...
  - Defeating public key exchange requires MITM attack
- Defenses against MITM
  - Keying material verification
    - Third-party authentication service (we know this won't work)
    - Out-of-band fingerprint exchange
    - Short authentication string
  - Key continuity
    - Verify that the same key is used for each call

Key Continuity

- Memorize keying material on first call to Bob
  - Generate an error/warning on any change
- False positives
  - Users change browsers regularly
  - This will generate a lot of errors (warning fatigue)
- False negatives
  - Remember, application is under control of the server
  - Application says it is calling `B0b` instead of `Bob`
    - Looks like a call to a new peer, not a changed key

Short Authentication String/Key Fingerprints

- Fingerprint: out-of-band exchange of hash of peer's key
  - Secure but requires out-of-band secure channel
- SAS: compute shared value from key exchange; read over voice channel
  - Susceptible to impersonation/voice conversion attacks [KM01, FEH]
  - Doesn't work with unknown speakers
- Both schemes rely on users checking
  - Which they won't [WT99]
- No known good way to prevent MITM by the calling service for average users

References

[Bar10a] Adam Barth. Prompting the user is security failure. RTC Workshop, 2010.

[Bar10b] Adam Barth. The Web Origin Concept. draft-ietf-websec-origin-00.txt, dec 2010.

[BWS09] Adam Barth, Joel Weinberger, and Dawn Song. Cross-Origin JavaScript Capability Leaks: Detection, Exploitation, and Defense. In Fabian Montrose, editor, *In Proc. of the 18th USENIX Security Symposium (USENIX Security 2009)*, August 2009.

[FEH] Mireia Farrs, Daniel Erro, and Javier Hern. Speaker recognition robustness to voice conversion.

[Fet11] Ian Fette. The WebSocket protocol. `draft-ietf-hybi-thewebsocketprotocol-06.txt`, February 2011.

[HCB+10] Lin-Shung Huang, Eric Y. Chen, Adam Barth, Eric Rescorla, and Collin Jackson. Transparent Proxies: Threat or Menace, 2010. In submission.

[JB08] Collin Jackson and Adam Barth. Beware of finer-grained origins. In *In Web 2.0 Security and Privacy (W2SP 2008)*, 2008.

[KM01] A. Kain and M. Macon. Design and Evaluation of a Voice Conversion Algorithm based on Spectral Envelope Mapping and Residual Prediction. Proceedings of ICASSP, May 2001.

[Ros10] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245, 2010.

[SDOF07] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. Emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *In Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[SEA+09] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness". Proceedings of the 18th USENIX Security Symposium, 2009.

[vK10] Anne van Kesteren. Cross-Origin Resource Sharing. `http://www.w3.org/TR/access-control/`, 2010.

[WT99] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.