RTC-Web
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2013

E. Rescorla
RTFM, Inc.

January 22, 2013

Security Considerations for RTC-Web
draft-ietf-rtcweb-security-04

Abstract

The Real-Time Communications on the Web (RTC-Web) working group is tasked with standardizing protocols for real-time communications between Web browsers. The major use cases for RTC-Web technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems (e.g., SIP-based soft phones) RTC-Web communications are directly controlled by some Web server, which poses new security challenges. For instance, a Web browser might expose a JavaScript API which allows a server to place a video call. Unrestricted access to such an API would allow any site which a user visited to "bug" a user's computer, capturing any activity which passed in front of their camera. This document defines the RTC-Web threat model and defines an architecture which provides security within that threat model.

Legal

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Real-Time Communications on the Web (RTC-Web) working group is tasked with standardizing protocols for real-time communications between Web browsers. The major use cases for RTC-Web technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems, (e.g., SIP-based[RFC3261] soft phones) RTC-Web communications are directly controlled by some Web server. A simple case is shown below.

```
                               +----------------+
                               |                |
                               |   Web Server   |
                               |                |
                               +----------------+
                                   ^        ^
                                  /          \
                          HTTP   /            \   HTTP
                                /              \
                               /                \
                              v                  v
                           JS API              JS API
                     +-----------+            +-----------+
                     |           |    Media   |           |
                     |  Browser  |<---------->|  Browser  |
                     |           |            |           |
                     +-----------+            +-----------+
```

Figure 1: A simple RTC-Web system

In the system shown in Figure 1, Alice and Bob both have RTC-Web enabled browsers and they visit some Web server which operates a calling service. Each of their browsers exposes standardized JavaScript calling APIs (implementated as browser built-ins) which are used by the Web server to set up a call between Alice and Bob. While this system is topologically similar to a conventional SIP-based system (with the Web server acting as the signaling service and browsers acting as softphones), control has moved to the central Web server; the browser simply provides API points that are used by the calling service. As with any Web application, the Web server can move logic between the server and JavaScript in the browser, but regardless of where the code is executing, it is ultimately under control of the server.

It should be immediately apparent that this type of system poses new security challenges beyond those of a conventional VoIP system. In particular, it needs to contend with malicious calling services. For example, if the calling service can cause the browser to make a call

at any time to any callee of its choice, then this facility can be used to bug a user's computer without their knowledge, simply by placing a call to some recording service. More subtly, if the exposed APIs allow the server to instruct the browser to send arbitrary content, then they can be used to bypass firewalls or mount denial of service attacks. Any successful system will need to be resistant to this and other attacks.

A companion document [I-D.ietf-rtcweb-security-arch] describes a security architecture intended to address the issues raised in this document.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. The Browser Threat Model

The security requirements for RTC-Web follow directly from the requirement that the browser's job is to protect the user. Huang et al. [huang-w2sp] summarize the core browser security guarantee as:

> Users can safely visit arbitrary web sites and execute scripts provided by those sites.

It is important to realize that this includes sites hosting arbitrary malicious scripts. The motivation for this requirement is simple: it is trivial for attackers to divert users to sites of their choice. For instance, an attacker can purchase display advertisements which direct the user (either automatically or via user clicking) to their site, at which point the browser will execute the attacker's scripts. Thus, it is important that it be safe to view arbitrarily malicious pages. Of course, browsers inevitably have bugs which cause them to fall short of this goal, but any new RTC-Web functionality must be designed with the intent to meet this standard. The remainder of this section provides more background on the existing Web security model.

In this model, then, the browser acts as a TRUSTED COMPUTING BASE (TCB) both from the user's perspective and to some extent from the server's. While HTML and JS provided by the server can cause the browser to execute a variety of actions, those scripts operate in a sandbox that isolates them both from the user's computer and from each other, as detailed below.

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.

3.1. Access to Local Resources

While the browser has access to local resources such as keying material, files, the camera and the microphone, it strictly limits or forbids web servers from accessing those same resources. For instance, while it is possible to produce an HTML form which will allow file upload, a script cannot do so without user consent and in fact cannot even suggest a specific file (e.g., /etc/passwd); the user must explicitly select the file and consent to its upload. [Note: in many cases browsers are explicitly designed to avoid dialogs with the semantics of "click here to screw yourself", as extensive research shows that users are prone to consent under such circumstances.]

Similarly, while Flash SWFs can access the camera and microphone, they explicitly require that the user consent to that access. In addition, some resources simply cannot be accessed from the browser at all. For instance, there is no real way to run specific executables directly from a script (though the user can of course be induced to download executable files and run them).

3.2. Same Origin Policy

Many other resources are accessible but isolated. For instance, while scripts are allowed to make HTTP requests via the XMLHttpRequest() API those requests are not allowed to be made to any server, but rather solely to the same ORIGIN from whence the script came.[RFC6454] (although CORS [CORS] and WebSockets [RFC6455] provides a escape hatch from this restriction, as described below.) This SAME ORIGIN POLICY (SOP) prevents server A from mounting attacks on server B via the user's browser, which protects both the user (e.g., from misuse of his credentials) and the server (e.g., from DoS attack).

More generally, SOP forces scripts from each site to run in their own, isolated, sandboxes. While there are techniques to allow them to interact, those interactions generally must be mutually consensual (by each site) and are limited to certain channels. For instance, multiple pages/browser panes from the same origin can read each

other's JS variables, but pages from the different origins--or even iframes from different origins on the same page--cannot.

3.3. Bypassing SOP: CORS, WebSockets, and consent to communicate

While SOP serves an important security function, it also makes it inconvenient to write certain classes of applications. In particular, mash-ups, in which a script from origin A uses resources from origin B, can only be achieved via a certain amount of hackery. The W3C Cross-Origin Resource Sharing (CORS) spec [CORS] is a response to this demand. In CORS, when a script from origin A executes what would otherwise be a forbidden cross-origin request, the browser instead contacts the target server to determine whether it is willing to allow cross-origin requests from A. If it is so willing, the browser then allows the request. This consent verification process is designed to safely allow cross-origin requests.

While CORS is designed to allow cross-origin HTTP requests, WebSockets [RFC6455] allows cross-origin establishment of transparent channels. Once a WebSockets connection has been established from a script to a site, the script can exchange any traffic it likes without being required to frame it as a series of HTTP request/ response transactions. As with CORS, a WebSockets transaction starts with a consent verification stage to avoid allowing scripts to simply send arbitrary data to another origin.

While consent verification is conceptually simple--just do a handshake before you start exchanging the real data--experience has shown that designing a correct consent verification system is difficult. In particular, Huang et al. [huang-w2sp] have shown vulnerabilities in the existing Java and Flash consent verification techniques and in a simplified version of the WebSockets handshake. In particular, it is important to be wary of CROSS-PROTOCOL attacks in which the attacking script generates traffic which is acceptable to some non-Web protocol state machine. In order to resist this form of attack, WebSockets incorporates a masking technique intended to randomize the bits on the wire, thus making it more difficult to generate traffic which resembles a given protocol.

4. Security for RTC-Web Applications

4.1. Access to Local Devices

As discussed in Section 1, allowing arbitrary sites to initiate calls violates the core Web security guarantee; without some access restrictions on local devices, any malicious site could simply bug a

user. At minimum, then, it MUST NOT be possible for arbitrary sites to initiate calls to arbitrary locations without user consent. This immediately raises the question, however, of what should be the scope of user consent.

In order for the user to make an intelligent decision about whether to allow a call (and hence his camera and microphone input to be routed somewhere), he must understand either who is requesting access, where the media is going, or both. As detailed below, there are two basic conceptual models:

> You are sending your media to entity A because you want to talk to Entity A (e.g., your mother).
> Entity A (e.g., a calling service) asks to access the user's devices with the assurance that it will transfer the media to entity B (e.g., your mother)

In either case, identity is at the heart of any consent decision. Moreover, identity is all that the browser can meaningfully enforce; if you are calling A, A can simply forward the media to C. Similarly, if you authorize A to place a call to B, A can call C instead. In either case, all the browser is able to do is verify and check authorization for whoever is controlling where the media goes. The target of the media can of course advertise a security/privacy policy, but this is not something that the browser can enforce. Even so, there are a variety of different consent scenarios that motivate different technical consent mechanisms. We discuss these mechanisms in the sections below.

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.

4.1.1. Calling Scenarios and User Expectations

While a large number of possible calling scenarios are possible, the scenarios discussed in this section illustrate many of the difficulties of identifying the relevant scope of consent.

4.1.1.1. Dedicated Calling Services

The first scenario we consider is a dedicated calling service. In this case, the user has a relationship with a calling site and repeatedly makes calls on it. It is likely that rather than having to give permission for each call that the user will want to give the calling service long-term access to the camera and microphone. This is a natural fit for a long-term consent mechanism (e.g., installing an app store "application" to indicate permission for the calling service.) A variant of the dedicated calling service is a gaming site (e.g., a poker site) which hosts a dedicated calling service to allow players to call each other.

With any kind of service where the user may use the same service to talk to many different people, there is a question about whether the user can know who they are talking to. If I grant permission to calling service A to make calls on my behalf, then I am implicitly granting it permission to bug my computer whenever it wants. This suggests another consent model in which a site is authorized to make calls but only to certain target entities (identified via media-plane cryptographic mechanisms as described in Section 4.3.2 and especially Section 4.3.2.3.) Note that the question of consent here is related to but distinct from the question of peer identity: I might be willing to allow a calling site to in general initiate calls on my behalf but still have some calls via that site where I can be sure that the site is not listening in.

4.1.1.2. Calling the Site You're On

Another simple scenario is calling the site you're actually visiting. The paradigmatic case here is the "click here to talk to a representative" windows that appear on many shopping sites. In this case, the user's expectation is that they are calling the site they're actually visiting. However, it is unlikely that they want to provide a general consent to such a site; just because I want some information on a car doesn't mean that I want the car manufacturer to be able to activate my microphone whenever they please. Thus, this suggests the need for a second consent mechanism where I only grant consent for the duration of a given call. As described in Section 3.1, great care must be taken in the design of this interface to avoid the users just clicking through. Note also that the user interface chrome must clearly display elements showing that the call is continuing in order to avoid attacks where the calling site just leaves it up indefinitely but shows a Web UI that implies otherwise.

4.1.1.3. Calling to an Ad Target

In both of the previous cases, the user has a direct relationship (though perhaps a transient one) with the target of the call. Moreover, in both cases he is actually visiting the site of the person he is being asked to trust. However, this is not always so. Consider the case where a user is a visiting a content site which hosts an advertisement with an invitation to call for more information. When the user clicks the ad, they are connected with the advertiser or their agent.

The relationships here are far more complicated: the site the user is actually visiting has no direct relationship with the advertiser; they are just hosting ads from an ad network. The user has no relationship with the ad network, but desires one with the advertiser, at least for long enough to learn about their products. At minimum, then, whatever consent dialog is shown needs to allow the user to have some idea of the organization that they are actually calling.

However, because the user also has some relationship with the hosting site, it is also arguable that the hosting site should be allowed to express an opinion (e.g., to be able to allow or forbid a call) since a bad experience with an advertiser reflect negatively on the hosting site [this idea was suggested by Adam Barth]. However, this obviously presents a privacy challenge, as sites which host advertisements in IFRAMEs often learn very little about whether individual users clicked through to the ads, or even which ads were presented.

4.1.2. Origin-Based Security

Now that we have seen another use case, we can start to reason about the security requirements.

As discussed in Section 3.2, the basic unit of Web sandboxing is the origin, and so it is natural to scope consent to origin. Specifically, a script from origin A MUST only be allowed to initiate communications (and hence to access camera and microphone) if the user has specifically authorized access for that origin. It is of course technically possible to have coarser-scoped permissions, but because the Web model is scoped to origin, this creates a difficult mismatch.

Arguably, origin is not fine-grained enough. Consider the situation where Alice visits a site and authorizes it to make a single call. If consent is expressed solely in terms of origin, then at any future visit to that site (including one induced via mash-up or ad network),

the site can bug Alice's computer, use the computer to place bogus calls, etc. While in principle Alice could grant and then revoke the privilege, in practice privileges accumulate; if we are concerned about this attack, something else is needed. There are a number of potential countermeasures to this sort of issue.

Individual Consent
   Ask the user for permission for each call.

Callee-oriented Consent
   Only allow calls to a given user.

Cryptographic Consent
   Only allow calls to a given set of peer keying material or to a cryptographically established identity.

Unfortunately, none of these approaches is satisfactory for all cases. As discussed above, individual consent puts the user's approval in the UI flow for every call. Not only does this quickly become annoying but it can train the user to simply click "OK", at which point the consent becomes useless. Thus, while it may be necessary to have individual consent in some case, this is not a suitable solution for (for instance) the calling service case. Where necessary, in-flow user interfaces must be carefully designed to avoid the risk of the user blindly clicking through.

The other two options are designed to restrict calls to a given target. Callee-oriented consent provided by the calling site not work well because a malicious site can claim that the user is calling any user of his choice. One fix for this is to tie calls to a cryptographically established identity. While not suitable for all cases, this approach may be useful for some. If we consider the advertising case described in Section 4.1.1.3, it's not particularly convenient to require the advertiser to instantiate an iframe on the hosting site just to get permission; a more convenient approach is to cryptographically tie the advertiser's certificate to the communication directly. We're still tying permissions to origin here, but to the media origin (and-or destination) rather than to the Web origin. [I-D.ietf-rtcweb-security-arch] and [I-D.rescorla-rtcweb-generic-idp] describe mechanisms which facilitate this sort of consent.

Another case where media-level cryptographic identity makes sense is when a user really does not trust the calling site. For instance, I might be worried that the calling service will attempt to bug my computer, but I also want to be able to conveniently call my friends. If consent is tied to particular communications endpoints, then my risk is limited. Naturally, it is somewhat challenging to design UI

primitives which express this sort of policy. The problem becomes even more challenging in multi-user calling cases.

4.1.3. Security Properties of the Calling Page

Origin-based security is intended to secure against web attackers. However, we must also consider the case of network attackers. Consider the case where I have granted permission to a calling service by an origin that has the HTTP scheme, e.g., http://calling-service.example.com. If I ever use my computer on an unsecured network (e.g., a hotspot or if my own home wireless network is insecure), and browse any HTTP site, then an attacker can bug my computer. The attack proceeds like this:

1. I connect to http://anything.example.org/. Note that this site is unaffiliated with the calling service.
2. The attacker modifies my HTTP connection to inject an IFRAME (or a redirect) to http://calling-service.example.com
3. The attacker forges the response apparently http://calling-service.example.com/ to inject JS to initiate a call to himself.

Note that this attack does not depend on the media being insecure. Because the call is to the attacker, it is also encrypted to him. Moreover, it need not be executed immediately; the attacker can "infect" the origin semi-permanently (e.g., with a web worker or a popunder) and thus be able to bug me long after I have left the infected network. This risk is created by allowing calls at all from a page fetched over HTTP.

Even if calls are only possible from HTTPS sites, if the site embeds active content (e.g., JavaScript) that is fetched over HTTP or from an untrusted site, because that JavaScript is executed in the security context of the page [finer-grained]. Thus, it is also dangerous to allow RTC-Web functionality from HTTPS origins that embed mixed content. Note: this issue is not restricted to PAGES which contain mixed content. If a page from a given origin ever loads mixed content then it is possible for a network attacker to infect the browser's notion of that origin semi-permanently.

4.2. Communications Consent Verification

As discussed in Section 3.3, allowing web applications unrestricted network access via the browser introduces the risk of using the browser as an attack platform against machines which would not otherwise be accessible to the malicious site, for instance because they are topologically restricted (e.g., behind a firewall or NAT). In order to prevent this form of attack as well as cross-protocol

attacks it is important to require that the target of traffic explicitly consent to receiving the traffic in question. Until that consent has been verified for a given endpoint, traffic other than the consent handshake MUST NOT be sent to that endpoint.

4.2.1. ICE

Verifying receiver consent requires some sort of explicit handshake, but conveniently we already need one in order to do NAT hole-punching. ICE [RFC5245] includes a handshake designed to verify that the receiving element wishes to receive traffic from the sender. 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). In order to achieve this objective with ICE, the STUN transaction IDs must be generated by the browser and MUST NOT be made available to the initiating script, even via a diagnostic interface. Verifying receiver consent also requires verifying the receiver wants to receive traffic from a particular sender, and at this time; for example a malicious site may simply attempt ICE to known servers that are using ICE for other sessions. ICE provides this verification as well, by using the STUN credentials as a form of per-session shared secret. Those credentials are known to the Web application, but would need to also be known and used by the STUN-receiving element to be useful.

There also needs to be some mechanism for the browser to verify that the target of the traffic continues to wish to receive it. Obviously, some ICE-based mechanism will work here, but it has been observed that because ICE keepalives are indications, they will not work here, so some other mechanism is needed.

[[OPEN ISSUE: Do we need some way of verifying the expected traffic rate, not just consent to receive traffic at all.]]

4.2.2. Masking

Once consent is verified, there still is some concern about misinterpretation attacks as described by Huang et al.[huang-w2sp]. As long as communication is limited to UDP, then this risk is probably limited, thus masking is not required for UDP. I.e., once communications consent has been verified, it is most likely safe to allow the implementation to send arbitrary UDP traffic to the chosen destination, provided that the STUN keepalives continue to succeed. In particular, this is true for the data channel if DTLS is used because DTLS (with the anti-chosen plaintext mechanisms required by TLS 1.1) does not allow the attacker to generate predictable

ciphertext. However, with TCP the risk of transparent proxies becomes much more severe. If TCP is to be used, then WebSockets style masking MUST be employed. [Note: current thinking in the RTCWEB WG is not to support TCP and to support SCTP over DTLS, thus removing the need for masking.]

4.2.3. Backward Compatibility

A requirement to use ICE limits compatibility with legacy non-ICE clients. It seems unsafe to completely remove the requirement for some check. All proposed checks have the common feature that the browser sends some message to the candidate traffic recipient and refuses to send other traffic until that message has been replied to. The message/reply pair must be generated in such a way that an attacker who controls the Web application cannot forge them, generally by having the message contain some secret value that must be incorporated (e.g., echoed, hashed into, etc.). Non-ICE candidates for this role (in cases where the legacy endpoint has a public address) include:

- o STUN checks without using ICE (i.e., the non-RTC-web endpoint sets up a STUN responder.)
- o Use or RTCP as an implicit reachability check.

In the RTCP approach, the RTC-Web endpoint is allowed to send a limited number of RTP packets prior to receiving consent. This allows a short window of attack. In addition, some legacy endpoints do not support RTCP, so this is a much more expensive solution for such endpoints, for which it would likely be easier to implement ICE. For these two reasons, an RTCP-based approach does not seem to address the security issue satisfactorily.

In the STUN approach, the RTC-Web endpoint is able to verify that the recipient is running some kind of STUN endpoint but unless the STUN responder is integrated with the ICE username/password establishment system, the RTC-Web endpoint cannot verify that the recipient consents to this particular call. This may be an issue if existing STUN servers are operated at addresses that are not able to handle bandwidth-based attacks. Thus, this approach does not seem satisfactory either.

If the systems are tightly integrated (i.e., the STUN endpoint responds with responses authenticated with ICE credentials) then this issue does not exist. However, such a design is very close to an ICE-Lite implementation (indeed, arguably is one). An intermediate approach would be to have a STUN extension that indicated that one was responding to RTC-Web checks but not computing integrity checks based on the ICE credentials. This would allow the use of standalone

STUN servers without the risk of confusing them with legacy STUN servers. If a non-ICE legacy solution is needed, then this is probably the best choice.

Once initial consent is verified, we also need to verify continuing consent, in order to avoid attacks where two people briefly share an IP (e.g., behind a NAT in an Internet cafe) and the attacker arranges for a large, unstoppable, traffic flow to the network and then leaves. The appropriate technologies here are fairly similar to those for initial consent, though are perhaps weaker since the threats is less severe.

4.2.4. IP Location Privacy

Note that as soon as the callee sends their ICE candidates, the caller learns the callee's IP addresses. The callee's server reflexive address reveals a lot of information about the callee's location. In order to avoid tracking, implementations may wish to suppress the start of ICE negotiation until the callee has answered. In addition, either side may wish to hide their location entirely by forcing all traffic through a TURN server.

4.3. Communications Security

Finally, we consider a problem familiar from the SIP world: communications security. For obvious reasons, it MUST be possible for the communicating parties to establish a channel which is secure against both message recovery and message modification. (See [RFC5479] for more details.) This service must be provided for both data and voice/video. Ideally the same security mechanisms would be used for both types of content. Technology for providing this service (for instance, DTLS [RFC4347] and DTLS-SRTP [RFC5763]) is well understood. However, we must examine this technology to the RTC-Web context, where the threat model is somewhat different.

In general, it is important to understand that unlike a conventional SIP proxy, the calling service (i.e., the Web server) controls not only the channel between the communicating endpoints but also the application running on the user's browser. While in principle it is possible for the browser to cut the calling service out of the loop and directly present trusted information (and perhaps get consent), practice in modern browsers is to avoid this whenever possible. "In-flow" modal dialogs which require the user to consent to specific actions are particularly disfavored as human factors research indicates that unless they are made extremely invasive, users simply agree to them without actually consciously giving consent. [abarth-rtcweb]. Thus, nearly all the UI will necessarily be rendered by the browser but under control of the calling service.

This likely includes the peer's identity information, which, after all, is only meaningful in the context of some calling service.

This limitation does not mean that preventing attack by the calling service is completely hopeless. However, we need to distinguish between two classes of attack:

Retrospective compromise of calling service.
The calling service is is non-malicious during a call but subsequently is compromised and wishes to attack an older call.

During-call attack by calling service.
The calling service is compromised during the call it wishes to attack.

Providing security against the former type of attack is practical using the techniques discussed in Section 4.3.1. However, it is extremely difficult to prevent a trusted but malicious calling service from actively attacking a user's calls, either by mounting a MITM attack or by diverting them entirely. (Note that this attack applies equally to a network attacker if communications to the calling service are not secured.) We discuss some potential approaches and why they are likely to be impractical in Section 4.3.2.

4.3.1. Protecting Against Retrospective Compromise

In a retrospective attack, the calling service was uncompromised during the call, but that an attacker subsequently wants to recover the content of the call. We assume that the attacker has access to the protected media stream as well as having full control of the calling service.

If the calling service has access to the traffic keying material (as in SDES [RFC4568]), then retrospective attack is trivial. This form of attack is particularly serious in the Web context because it is standard practice in Web services to run extensive logging and monitoring. Thus, it is highly likely that if the traffic key is part of any HTTP request it will be logged somewhere and thus subject to subsequent compromise. It is this consideration that makes an automatic, public key-based key exchange mechanism imperative for RTC-Web (this is a good idea for any communications security system) and this mechanism SHOULD provide perfect forward secrecy (PFS). The signaling channel/calling service can be used to authenticate this mechanism.

In addition, the system MUST NOT provide any APIs to extract either long-term keying material or to directly access any stored traffic

keys. Otherwise, an attacker who subsequently compromised the calling service might be able to use those APIs to recover the traffic keys and thus compromise the traffic.

4.3.2. Protecting Against During-Call Attack

Protecting against attacks during a call is a more difficult proposition. Even if the calling service cannot directly access keying material (as recommended in the previous section), it can simply mount a man-in-the-middle attack on the connection, telling Alice that she is calling Bob and Bob that he is calling Alice, while in fact the calling service is acting as a calling bridge and capturing all the traffic. While in theory it is possible to construct techniques which protect against this form of attack, in practice these techniques all require far too much user intervention to be practical, given the user interface constraints described in [abarth-rtcweb].

4.3.2.1. Key Continuity

One natural approach is to use "key continuity". While a malicious calling service can present any identity it chooses to the user, it cannot produce a private key that maps to a given public key. Thus, it is possible for the browser to note a given user's public key and generate an alarm whenever that user's key changes. SSH [RFC4251] uses a similar technique. (Note that the need to avoid explicit user consent on every call precludes the browser requiring an immediate manual check of the peer's key).

Unfortunately, this sort of key continuity mechanism is far less useful in the RTC-Web context. First, much of the virtue of RTC-Web (and any Web application) is that it is not bound to particular piece of client software. Thus, it will be not only possible but routine for a user to use multiple browsers on different computers which will of course have different keying material (SACRED [RFC3760] notwithstanding.) Thus, users will frequently be alerted to key mismatches which are in fact completely legitimate, with the result that they are trained to simply click through them. As it is known that users routinely will click through far more dire warnings [cranor-wolf], it seems extremely unlikely that any key continuity mechanism will be effective rather than simply annoying.

Moreover, it is trivial to bypass even this kind of mechanism. Recall that unlike the case of SSH, the browser never directly gets the peer's identity from the user. Rather, it is provided by the calling service. Even enabling a mechanism of this type would require an API to allow the calling service to tell the browser "this is a call to user X". All the calling service needs to do to avoid

triggering a key continuity warning is to tell the browser that "this is a call to user Y" where Y is close to X. Even if the user actually checks the other side's name (which all available evidence indicates is unlikely), this would require (a) the browser to trusted UI to provide the name and (b) the user to not be fooled by similar appearing names.

4.3.2.2. Short Authentication Strings

ZRTP [RFC6189] uses a "short authentication string" (SAS) which is derived from the key agreement protocol. This SAS is designed to be read over the voice channel and if confirmed by both sides precludes MITM attack. The intention is that the SAS is used once and then key continuity (though a different mechanism from that discussed above) is used thereafter.

Unfortunately, the SAS does not offer a practical solution to the problem of a compromised calling service. "Voice conversion" systems, which modify voice from one speaker to make it sound like another, are an active area of research. These systems are already good enough to fool both automatic recognition systems [farus-conversion] and humans [kain-conversion] in many cases, and are of course likely to improve in future, especially in an environment where the user just wants to get on with the phone call. Thus, even if SAS is effective today, it is likely not to be so for much longer. Moreover, it is possible for an attacker who controls the browser to allow the SAS to succeed and then simulate call failure and reconnect, trusting that the user will not notice that the "no SAS" indicator has been set (which seems likely).

Even were SAS secure if used, it seems exceedingly unlikely that users will actually use it. As discussed above, the browser UI constraints preclude requiring the SAS exchange prior to completing the call and so it must be voluntary; at most the browser will provide some UI indicator that the SAS has not yet been checked. However, it it is well-known that when faced with optional mechanisms such as fingerprints, users simply do not check them [whitten-johnny] Thus, it is highly unlikely that users will ever perform the SAS exchange.

Once uses have checked the SAS once, key continuity is required to avoid them needing to check it on every call. However, this is problematic for reasons indicated in Section 4.3.2.1. In principle it is of course possible to render a different UI element to indicate that calls are using an unauthenticated set of keying material (recall that the attacker can just present a slightly different name so that the attack shows the same UI as a call to a new device or to someone you haven't called before) but as a practical matter, users

simply ignore such indicators even in the rather more dire case of mixed content warnings.

Despite these difficulties, users should be afforded an opportunity to view an SAS or fingerprint where available, as it is the only mechanism for the user to directly verify the peer's identity without trusting any third party identity system (assuming, of course, that they trust their own software).

4.3.2.3. Third Party Identity

The conventional approach to providing communications identity has of course been to have some third party identity system (e.g., PKI) to authenticate the endpoints. Such mechanisms have proven to be too cumbersome for use by typical users (and nearly too cumbersome for administrators). However, a new generation of Web-based identity providers (BrowserID, Federated Google Login, Facebook Connect, OAuth, OpenID, WebFinger), has recently been developed and use Web technologies to provide lightweight (from the user's perspective) third-party authenticated transactions. It is possible (see [I-D.rescorla-rtcweb-generic-idp]) to use systems of this type to authenticate RTCWEB calls, linking them to existing user notions of identity (e.g., Facebook adjacencies). Specifically, the third-party identity system is used to bind the user's identity to cryptographic keying material which is then used to authenticate the calling endpoints. Calls which are authenticated in this fashion are naturally resistant even to active MITM attack by the calling site.

Note that there is one special case in which PKI-style certificates do provide a practical solution: calls from end-users to large sites. For instance, if you are making a call to Amazon.com, then Amazon can easily get a certificate to authenticate their media traffic, just as they get one to authenticate their Web traffic. This does not provide additional security value in cases in which the calling site and the media peer are one in the same, but might be useful in cases in which third parties (e.g., ad networks or retailers) arrange for calls but do not participate in them.

4.3.2.4. Page Access to Media

Identifying the identity of the far media endpoint is a necessary but not sufficient condition for providing media security. In RTCWEB, media flows are rendered into HTML5 MediaStreams which can be manipulated by the calling site. Obviously, if the site can modify or view the media, then the user is not getting the level of assurance they would expect from being able to authenticate their peer. In many cases, this is acceptable because the user values site-based special effects over complete security from the site.

However, there are also cases where users wish to know that the site cannot interfere. In order to facilitate that, it will be necessary to provide features whereby the site can verifiably give up access to the media streams. This verification must be possible both from the local side and the remote side. I.e., I must be able to verify that the person I am calling has engaged a secure media mode. In order to achieve this it will be necessary to cryptographically bind an indication of the local media access policy into the cryptographic authentication procedures detailed in the previous sections.

5. Security Considerations

This entire document is about security.

6. Acknowledgements

Bernard Aboba, Harald Alvestrand, Dan Druta, Cullen Jennings, Hadriel Kaplan (S 4.2.1), Matthew Kaufman, Martin Thomson, Magnus Westerland.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[CORS] van Kesteren, A., "Cross-Origin Resource Sharing".

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "RTCWEB Security Architecture", draft-ietf-rtcweb-security-arch-05 (work in progress), October 2012.

[I-D.kaufman-rtcweb-security-ui]
Kaufman, M., "Client Security User Interface Requirements for RTCWEB", draft-kaufman-rtcweb-security-ui-00 (work in progress), June 2011.

[I-D.rescorla-rtcweb-generic-idp]
Rescorla, E., "RTCWEB Generic Identity Provider Interface", draft-rescorla-rtcweb-generic-idp-01 (work in progress), March 2012.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3760] Gustafson, D., Just, M., and M. Nystrom, "Securely Available Credentials (SACRED) - Credential Server Framework", RFC 3760, April 2004.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5479] Wing, D., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, April 2009.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC6189] Zimmermann, P., Johnston, A., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, April 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[abarth-rtcweb]
Barth, A., "Prompting the user is security failure", RTC-Web Workshop.

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

[farus-conversion]
Farrus, M., Erro, D., and J. Hernando, "Speaker Recognition Robustness to Voice Conversion".

[finer-grained]
Barth, A. and C. Jackson, "Beware of Finer-Grained Origins", W2SP, 2008.

[huang-w2sp]
Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C. Jackson, "Talking to Yourself for Fun and Profit", W2SP, 2011.

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

[whitten-johnny]
Whitten, A. and J. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0", Proceedings of the 8th USENIX Security Symposium, 1999.

Author's Address

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com