Truncating TLS Connections to Violate Beliefs in Web Applications

Ben Smyth and Alfredo Pironti

INRIA Paris-Rocquencourt, Paris, France

Abstract

We identify logical web application flaws which can be exploited by TLS truncation attacks to desynchronize the user- and server-perspective of an application's state. It follows immediately that servers may make false assumptions about users, hence, the flaw constitutes a security vulnerability. Moreover, in the context of authentication systems, we exploit the vulnerability to launch the following practical attacks: we exploit the Helios electronic voting system to cast votes on behalf of honest voters, take full control of Microsoft Live accounts, and gain temporary access to Google accounts.

Keywords. Attack, authentication, exploit, Google, Helios, logical flaw, Microsoft, sign-out, single sign-on, TLS truncation, web applications.

1 Introduction

Web applications are distributed, concurrent algorithms that are executed over public communication networks such as the Internet. Since the communication medium is public, there is a possibility of interference by a powerful adversary and this has made the design of secure web applications difficult [23]. Nevertheless, the burden is somewhat reduced by HTTPS.

HTTPS [17] uses HTTP [8] over TLS [6] to ensure *confidentiality* and *integrity* of a communication session, in particular, integrity ensures that all messages are received as sent. HTTPS is typically used by web applications to protect client-server communication. However, TLS alone is generally insufficient to protect sensitive operations (such as protecting authentication credentials, for example), since the security of such operations is also dependent upon the application logic and TLS provides no protection against logical application flaws. Moreover, TLS only ensures messages are received as sent for a *single* connection and, crucially, does not guarantee the ordering of messages in *multiple* connections; this is particularly pertinent to web applications, due to the use of parallel computation, whereby each simultaneous communication uses a different TLS connection (for example, browsers maintain multiple persistent connections to load content in parallel, thereby reducing latency). Accordingly, web application logic must protect sensitive operations in the presence of parallel computation.

In this paper, we highlight logical application flaws which permit desynchronisation between the application's state, as perceived by the user and the server, by truncating selected TLS connections. Furthermore, we exploit the desynchronisation to attack real systems.

1.1 Application state

In web applications, the user is typically notified of the server's state using some feedback mechanism [14, 15]. Feedback can be *positive*, when the user is notified of successful state changes, or *negative*, when the user is notified on errors. The absence of feedback can cause confusion – for instance, if a user attempts to save a file and no feedback is provided, then the user does not know if her file was saved or not – and violates basic design principles [13]. By comparison, we focus on incorrect feedback: web applications that generate (positive) user feedback *before* the server has committed to a state change.

1.2 Truncating TLS connections

TLS security guarantees are defined with respect to the following termination modes: *graceful* connection closure (that is, at the end of a successful connection) and *fatal* closure (that is, at the end of an unsuccessful connection, for example, after receiving a corrupt message). In the event of graceful closure, TLS ensures that *all* messages are received as sent, by comparison, in the event of fatal closure, TLS ensures that a *prefix* of all messages

[*] Smyth and Pironti are supported by the ERC Grant StG-2010 259639-CRYSP.

[†] This paper is supported by the following videos: [20, 21, 22].

are received as sent. At a lower level of abstraction, TLS is permitted to split messages into *fragments*, and TLS guarantees ordered delivery of all fragments on graceful closure and a prefix of all fragments on fatal closure.

Historically, graceful and fatal closure modes were not supported in SSL (TLS's predecessor) until version 3.0 [9]. Today, TLS termination modes are not distinguished by many applications, indeed, major browsers (including Internet Explorer, Firefox, Chrome and Safari) and HTTP servers do not always distinguish between graceful and fatal closure. As a consequence, the TLS specification notes *"failure to properly close a connection no longer requires that a session not be resumed [...] to conform with widespread implementation practice"* [6, §7.2.1]. Unfortunately, ignoring the termination mode can result in TLS truncation attacks [4]. Let us illustrate such a truncation attack with a simple example [5]. Suppose a user initiates a wire transfer to *Charlie's Angels* using the following HTTP request:

```
POST /wire_transfer.php HTTP/1.1
Host: mybank.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 40

amount=1000&recipient=Charlie%27s_Angels
```

Further suppose that the request is fragmented by TLS as follows: "`POST [...]  recipient=Charlie`" and "`%27s_Angels`". In this setting, an adversary can make a transfer to Charlie, rather than the intended recipient, simply by dropping the second (and crucially the last) fragment (e.g., by closing the underlying TCP connection after the first fragment is delivered). The attack works against HTTP servers that ignore the connection termination status and `Content-Length` field, in particular, a payload shorter than the specified length will be accepted. On the other hand, the attack can be thwarted by insisting that either the length of the payload is correct or wire transfers are only initiated upon graceful closure of the TLS connection. Henceforth, we shall consider a simpler class of truncation attacks: truncation attacks which drop messages, rather than fragments.

1.3 Adversary model

We assume that web applications should achieve their objectives in the presence of an adversary that has full control of the network; this assumption is standard in both the symbolic [7] and computational [10] cryptographic verification models and, furthermore, such power is typical of governments and ISPs [18, 19]. Since the adversary has complete control of the network, messages may be read, deleted, and injected. Moreover, the adversary is able to manipulate data contained within unencrypted messages. We also assume that the adversary has access to shared computers, but cannot compromise them; we motivate the practicality of this assumption in Section 1.4.

1.4 Contribution and structure

We identify logical web application flaws which can be exploited by TLS truncation attacks to ensure that a client receives feedback about a state change, whereas, the server is unaware of any goal to reach the aforementioned state. Consequently, servers may make false assumptions about users and, as discussed below, this flaw can be exploited to violate security objectives, hence, the flaw is a security vulnerability. We believe this insight constitutes a new attack vector.

Using our attack vector, we demonstrate the following practical attacks: we exploit the Helios electronic voting system to cast votes on behalf of honest voters (Section 2), take full control of Microsoft Live accounts (Section 3), and gain temporary access to Google accounts (Section 4). In essence, our attacks against Helios and Microsoft Live are due to the following application flaw: the sign-out procedure spawns several TLS connections such that one connection provides the user feedback and, independently, the others revoke the user's authentication from the servers. It follows that we can truncate the TLS connections which revoke authentication whilst notifying the user that the sign-out procedure successfully completed, hence, the server believes that the user is authenticated, but the user does not. In this context, an adversary can use the voting terminal after honest voters to cast votes on their behalf and, in the context of Microsoft Live, an adversary can take control of users' accounts, assuming the user shares a computer with the adversary (this setting is typically of computers in public libraries and work places, for example). We stress that physical access *after* the user believes she has signed out is sufficient and the attack does not rely on tampering with voting terminals nor compromising shared computers (e.g., using malware). By comparision, our attack against Google exploits an application flaw *and* poor handling of TLS termination modes, as we shall discuss in Section 4.3.

2 Attack I: Helios electronic voting system

Helios [2] is an open-source web-based end-to-end verifiable electronic voting system, which has been used in several binding elections, for instance, the International Association of Cryptologic Research (IACR) have used Helios annually since 2010 to elect its board members [12, 3, 11], the Catholic University of Louvain adopted the system to elect the university president [2], and Princeton University have used Helios to elect several student governments [1, 16]. In this section, we analyse the current Helios release and discover a flaw in the authentication logic. The flaw allows an adversary to drop sign-out requests (although the sign-out request is encrypted in the TLS traffic, it has a unique size, which allows such a request to be identified and dropped by the adversary), whilst providing the voter with positive feedback that their request succeeded. The adversary can then use the voting terminal to take control of the voter's session and cast a ballot on the voter's behalf. The attack works under the standard assumption that the adversary controls the network and may access the voting terminal, but, we stress, in accordance with the Helios threat model, the attack does not rely on modifying the voting terminal software.

2.1 Preliminaries

The trace in Figure 1 lists the requests performed by a browser during a normal ballot casting/sign-out procedure. The first request confirms the voter's intention to cast their ballot and the server responds with a redirect. The second request handles the redirect and the server responds with an HTML payload confirming receipt of the voter's ballot and provides the voter with the following notification: *"For your safety, we have logged you out."* The third request is the voter's sign-out request and the server responds with a redirect.

2.2 Our attack

By observation of the trace (Figure 1), we can immediately observe a flaw in the authentication logic: voters are given positive sign-out feedback *before* the voting terminal makes sign-out requests. This flaw can be exploited by a truncation attack which drops sign-out requests without the voters' knowledge. In this context, the voting terminal maintains a session with the server and the adversary can use the voting terminal to take control of voting sessions and, therefore, cast ballots on behalf of honest voters (Helios permits *re-voting* – that is, a voter may cast arbitrarily many ballots and only the last will be counted – hence, ballots cast by the adversary will overwrite honestly cast ballots). We stress that TLS does not offer protection against this kind of attack, because any TLS connection made by the adversary is independent from TLS connections used by voters and TLS does not guarantee the ordering of messages between connections, that is, dropping the third request does not prevent the adversary making a future connection to the Helios server and casting ballots on behalf of honest voters.

Launching the attack in practise. We have successfully demonstrated the attack in a supporting video [20]. For the purposes of this video demonstration, the voting terminal is modelled as a virtual machine running Ubuntu 10.04.4 LTS (Lucid Lynx) from an ISO image and the network is modelled as the host, which the adversary controls. In this setting, we used basic traffic analysis techniques to observe that request 3 (the sign-out request) is 701 bytes in length and we configured the host's *iptables* to drop packets of this length, using the following command:

```
iptables -A OUTPUT -m length --length 701 -j DROP
```

The above command instructs the host to drop any packet from the virtual machine of length 701. The rule ensures that request 3 is dropped and, hence, the user incorrectly believes that she has been signed out.

Vulnerability disclosure. Our preliminary findings where disclosed to the Helios developers on 4 June 2012, and the findings of this report were released on 21 September 2012. At the time of publication, the vulnerability has not been fixed.

2.3 Countermeasures

We propose the following solution: the three client actions in the trace (Figure 1) should be atomic. Specifically, on receipt of the voter's intention to cast their ballot, the server should sign the voter out and provide positive feedback of these actions. If implementing this solution is difficult, then the following interim solution could be adopted: the voter should be more precisely informed about the internal transition state of the server, in particular, the page returned by the *cast_done* request should inform the voter that a sign-out operation is in progress, rather than giving positive feedback about a successful sign-out. In addition, the server should provide positive feedback after the *sign-out* request. In this setting, if the adversary drops a *sign-out* request, then the voter perceives that a sign-out operation was in progress, but it was never successfully completed.

3 Attack II: Microsoft services

We demonstrate a flaw in Microsoft Live's authentication logic which can be exploited by an adversary to gain access to a user's account. Our assumptions are as follows: the user has at least two open sessions with Microsoft and the adversary controls the network and can access a computer shared with the user. We stress that the attack does not rely on tampering with the shared computer, and the adversary only needs access *after* the user believes she signed out, i.e., when the user is likely to leave the computer unattended. In essence, our attack works as follows. First, a user visits *Hotmail* and is redirected to *login.live.com* for authentication. Secondly, once the user has finished reading her mail, she visits *account.live.com* and a session is seamlessly authenticated by Microsoft Live single sign-on service. (In the supporting video, the chosen account lacks some basic information, so the user is automatically redirected to *account.live.com* before she can access her email.) Thirdly, the user makes a *sign-out*

Figure 1 Trace of the ballot casting/sign-out procedure in Helios

1. POST `https://vote.heliosvoting.org/helios/elections/≪id≫/cast_confirm`
   Response: 302 - Moved Temporarily
   Location[`https://vote.heliosvoting.org/helios/elections/≪id≫/cast_done`]
2. GET: `https://vote.heliosvoting.org/helios/elections/≪id≫/cast_done`
   Response: 200 - OK; HTML payload:

   ```
   ...
   <p><b>For your safety, we have logged you out.</b></p>
   <iframe border="0" src="/auth/logout" frameborder="0" height="0" width="0">
   </iframe>
   ...
   ```
3. GET: `https://vote.heliosvoting.org/auth/logout`
   Response: 302 - Moved Temporarily
   Location[`http://vote.heliosvoting.org/`]

request from Hotmail and the server responds by guiding the user's browser through a series of further requests which are intended to terminate each active session, in particular, this procedure should ensure that sessions with both Hotmail and *accounts.live.com* are terminated. However, we show that an adversary can selectively prevent the termination of active sessions, whilst giving the user feedback that her sign-out request succeeded. Finally, when the user leaves the shared computer, the adversary can use the computer to access the user's account information on *account.live.com* and can reset the user's password, for example.

3.1 Preliminaries

The trace in Figure 2 shows a normal sign-out procedure, listing the requests a browser performs following a user sign-out request (for brevity, we omit some details). The trace shows that signing-out is not a centralised action and each Microsoft service maintains some session information. Intuitively, it follows that the sign-out procedure is designed to terminate each active service and, as can be observed from the trace, the procedure uses HTTP redirects and HTML pages with embedded Javascript. More precisely, the Javascript loaded in steps 2 and 3 is intended to work as follows. The session with *account.live.com* is terminated in step 4, the session with *accountservices.msn.com* is terminated in step 5, and the session with *mail.live.com* is terminated in step 6. Finally, as a consequence of step 6, the browser loads the Micorosft Live home page.

Server misconfiguration. We observe that requests 6 and 7 are sent over HTTP, rather than HTTPS, which trivially permits attacks. The fix is trivial: all requests in the sign-out procedure must be sent over HTTPS. In the remainder of this section, we assume this fix has been applied and, nonetheless, demonstrate a further attack, which constitutes the main subject of this section.

3.2 Our attack

Microsoft's decentralised sign-out procedure is not secure, because steps 4-6 are independent of each other and an adversary can drop request 4 (the request that signs the user out of the *account.live.com* service), by truncating a TLS connection, to prevent the user's `account.live.com` session from terminating. The sign-out procedure will then proceed as normal, signing the user out from the remaining services, and displaying the *"You've signed out"* message. The adversary can then use the shared computing terminal to access *account.live.com* as the user, add a recovery email address to the user's account, and then reset the user's password. The password reset operation sends an email to the address supplied by the adversary, thereby allowing the adversary to take full control of the user's account.

Launching the attack in practise. We have successfully demonstrated the attack in a supporting video [22], using the setup described in Section 2.2. In advance of the attack, we used basic traffic analysis techniques to observe that request 4 (the request signing the user out from *account.live.com*) is between 474 and 506 bytes in length, and we configure the network to drop packets in this range, using the following command:

```
iptables -A OUTPUT -m length --length 474:506
  -j DROP
```

(We acknowledge that more sophisticated traffic analysis techniques would be better suited to identifying and dropping packets, but our simplistic approach is sufficient for our demonstration.)

Figure 2 Trace of the sign-out procedure for Microsoft Live

1. GET `https://login.live.com/logout.srf?ct=1364567198&rver=6.1.6206.0&lc=1033&id=64855&ru=http\%2F\%2Fbay171.mail.live.com\%2Fhandlers\%2FSignout.mvc\%3Fservice\%3DLive.Mail&mkt=en-fr`
   Response: 200 - OK. The HTML payload initiates the two following requests (the Javascript code defined by `Login_Core.js` and `Login_Alt.js` is obfuscated and we have not attempted to reverse engineer the code, since it is not necessary for our attack).
2. GET `https://secure.shared.live.com/~Live.SiteContent.ID/~17.0.11/~/~/~/~/js/Login_Core.js`
   Response: 200 - OK.
3. GET `https://secure.shared.live.com/~Live.SiteContent.ID/~17.0.11/~/~/~/~/js/Login_Alt.js`
   Response: 200 - OK.
4. GET `https://account.live.com/logout.aspx?ct=1364567254`
   Response: 200 - OK
5. GET `https://accountservices.msn.com/LogoutMSN.srf?ct=1364567254`
   Response: 200 - OK
6. GET `http://bay171.mail.live.com/handlers/Signout.mvc?service=Live.Mail&lc=1033`
   Response: 302 - Moved Temporarily,
   Location[`http://g.live.com/9ep9nmso/so-EN-US`]
7. GET `http://g.live.com/9ep9nmso/so-EN-US`
   Response: 301 - Moved Permanently,
   Location[`https://signout.live.com/content/dam/imp/surfaces/mail_signout/v7/mail/en-us.html`]
8. GET `https://signout.live.com/content/dam/imp/surfaces/mail_signout/v7/mail/en-us.html`
   Response: 200 - OK
   This page redirects to the Microsoft Live portal and provides the user with the following positive feedback, namely, *"You've signed out."*

Variants of our attack. As can be observed from our trace (Figure 2), we can truncate TLS connections which are used to sign the user out from other services, in particular, we have successfully attacked Hotmail and MSN.

Vulnerability disclosure. We disclosed our findings to Microsoft on 2 April 2013 and Microsoft provided (3 July 2013) the following response for publication.

> Microsoft would like to thank the authors for sharing their research with us and allowing us to evaluate this issue. This [attack] scenario relies on gaining physical access to the user's computer and the ability to man in the middle its network connection. This scenario can be mitigated by clearing the browser history and closing the browser. Microsoft takes this and all issues very seriously and is investigating defense in depth strategies that could further help to mitigate this issue. We will continue to take appropriate action to help protect customers.

We acknowledge that deleting cookies is sufficient to prevent this attack. However, deleting cookies places an unnecessary burden upon the user. Accordingly, we recommend that a server-side solution is deployed as part of Microsoft's "*defense in depth strategy*" and, at the time of publication, we are discussing our countermeasures with Microsoft.

3.3 Countermeasures

We recommend that all authentication is managed centrally – by *account.live.com*, for example – therefore avoiding all of the problems highlighted in this sections. In addition, particularly sensitive actions, such as adding recovery email addresses, should be password protected.

We acknowledge that our solution (above) may not be viable, for example, it may violate some privacy policy or may be unsuited to Microsoft's architecture, in particular, it will create additional traffic for the Account server. In this case, we recommend modifying the sign-out procedure to use a chain of HTTP redirects over TLS as depicted in Figure 3. In comparison with Microsoft's sign-out procedure, our solution never returns an HTML page containing Javascript code, instead, we force sequential actions by returning a redirect for each page that would have been previously loaded in parallel by Javascript. One may implement this solution using well-implemented and carefully monitored URL redirectors. We believe chaining a sequence of HTTP redirects over TLS is a secure way of implementing a decentralised sign-out procedure: if an arbitrary request or response is blocked, then the browser hangs (or displays an error message); thereby en-

Figure 3 Sketch of a countermeasure for the sign-out procedure in Microsoft Live

1. GET `https://login.live.com/≪signout≫`
   Response: 302 - Moved Temporarily, Location[`https://account.live.com/≪signout≫`]
2. GET `https://account.live.com/≪signout≫`
   Response: 302 - Moved Temporarily, Location[`https://accountservices.msn.com/≪signout≫`]
3. GET `https://accountservices.msn.com/≪signout≫`
   Response: 302 - Moved Temporarily, Location[`https://mail.live.com/≪signout≫`]
4. GET `https://mail.live.com/≪signout≫`
   Response: 302 - Moved Temporarily, Location[`https://signout.live.com`]

suring that the user is aware of any failure. Alternatively, the Javascript code could be revised to ensure sequential loading of URLs, reporting sign-out failures to the user.

4 Attack III: Google services

We demonstrate a flaw in Gmail's authentication logic which can be exploited to gain access to a user's email account. Our assumptions are as follows: the user has at least two open sessions with Google and the adversary controls the network and can access a computer shared with the user. In essence, our attack works as follows. First, a user visits *Gmail* and authenticates a session on a shared computer. Secondly, once the user has finished reading her mail, she visits *Search* and a session is seamlessly authenticated by Google's single sign-on service. Thirdly, the user makes a *sign-out* request and the server responds by guiding the user's browser through a series of further requests which are intended to terminate each active session, in particular, this procedure should ensure that the session with Gmail is terminated. However, we show that an adversary can selectively prevent the termination of active sessions, whilst giving the user feedback that her sign-out request succeeded. Finally, when the user leaves the shared computer, the adversary can use the computer to access the user's Gmail account.

4.1 Preliminaries

The trace in Figure 4 shows a normal sign-out procedure, listing the requests a browser performs following a user sign out request (for brevity, we omit some details). As can be observed from the trace, the procedure is not centralised and a combination of HTML pages with embedded Javascript and HTTP redirects are used to sign-out the user. More precisely, the procedure is intended to work as follows. First, the session with *Accounts* is terminated and the browser is redirected. Secondly, the redirect responds with an HTML page containing an image[1] and some Javascript code which is executed once the page has completely loaded all other content, in particular, the script will be executed after the image has been loaded. Thirdly, the browser requests the aforementioned image from the *Gmail* server and the server terminates the user's session before responding with the requested image and instructions to reset all local cookies. Having executed the `doRedirect()` Javascript function, the browser is redirected and the fourth & fifth actions are similar to the second & third. Finally, in the sixth step, the browser loads Google's home page.

Server misconfiguration. We observe that requests 2 and 4 are sent over HTTP, rather than HTTPS, which trivially permits attacks. For example, an adversary can replace `ils=mail,s.FR` with `ils=s.FR` in request 2, which effectively skips requests 2 and 3, and proceeds with request 4. Consequently, at the end of the sign-out procedure, the user will be presented with feedback (namely, the user will observe a *sign-in* button) that her sign-out request succeeded, whilst her Gmail session is still active. The fix is trivial: all requests in the sign-out procedure must be sent over HTTPS. In the remainder of this section, we assume this fix has been applied.

4.2 Our attack

Google's decentralised sign-out procedure is not secure, because an adversary can drop request 3 (the request for an image from *Gmail*) by truncating a TLS connection using a TCP reset, thereby preventing Gmail from terminating the user's session. The TCP reset will prevent the image from loading and the user's browser will display the alternative *"Sign Out"* text. Nonetheless, due to poor handling of TLS termination modes, the browser will consider all content to be loaded and will execute the `doRedirect()` Javascript function, which allows the browser to proceed without notifying the user that Gmail has failed to terminate the user's session. Moreover, the final request gives the user feedback that her sign-out request has succeeded. The adversary can then use the

[1] As an aside, we remark that this is an inefficient technique to make a dummy request to a server, since the image is a compressed gif pixel of 43 bytes, whereas an uncompressed image – in an alternative format – would be smaller in size.

Figure 4 Trace of the sign-out procedure for Google services

1. GET `https://accounts.google.com/Logout?continue=https://www.google.com/webhp`

   Response: 302 - Moved Temporarily, Location[`http://www.google.com/accounts/Logout2?ilo=1&ils=mail,s.FR&ilc=0&continue=https://www.google.com/webhp?zx=1388193849`]

2. GET `http://www.google.com/accounts/Logout2?ilo=1&ils=mail,s.FR&ilc=0&continue=https://www.google.com/webhp?zx=1388193849`

   Response: 200 - OK; HTML payload:

   ```
   <body onload="doRedirect()">
   <script type="text/javascript">
    function doRedirect() {
     location.replace("http://www.google.fr/accounts/Logout2?
       ilo=1&ils=s.FR&ilc=1&continue=https://www.google.com/webhp?zx=1076119961");
    }
   </script>
   <img width="0" height="0" alt="Sign Out"
     src="https://mail.google.com/mail?logout=img&zx=-2531125006460954395">
   </body>
   ```

3. GET `https://mail.google.com/mail?logout=img&zx=-2531125006460954395`

   Response: 200 - OK; a one pixel gif.

4. GET `http://www.google.fr/accounts/Logout2?ilo=1&ils=s.FR&ilc=1&continue=https://www.google.com/webhp?zx=1076119961`

   Response: 200 - OK; HTML payload:

   ```
   <body onload="doRedirect()">
   <script type="text/javascript">
    function doRedirect() {
     location.replace("https://www.google.com/webhp");
    }
   </script>
   <img width="0" height="0" alt="Sign Out"
     src="https://accounts.google.fr/accounts/ClearSID?zx=-1920517974">
   </body>
   ```

5. GET `https://accounts.google.fr/accounts/ClearSID?zx=-1920517974`

   Response: 200 - OK; a one pixel gif.

6. GET `https://www.google.com/webhp`

   Response: 200 - OK

shared computing terminal to access the user's Gmail account; access is limited to approximately five minutes, due to server-side housekeeping.

Launching the attack in practise. We have successfully demonstrated the attack in a supporting video [21], using the setup described in Section 2.2. (In the video we visit `google.it` before visiting Gmail, this step is not necessary.) In advance of the attack, we used basic traffic analysis techniques to observe that request 3 (the image request to `mail.google.com`) is between 1165 and 1195 bytes in length, and we configure the network to reject packets in this range, using the following command:

```
iptables -A OUTPUT -m length --length 1165:1195
  -p tcp -j REJECT --reject-with tcp-reset
```

The rule ensures that request 3 is dropped during Google's sign-out procedure and the TCP reset causes the browser to abort loading the image and immediately process the Javascript code, thus executing request 4.

Variants of our attack. As can be observed from our trace (Figure 4), we can truncate TLS connections which are used to fetch images and thus other Google services are also affected, in particular, we have successfully attacked Search and YouTube.

Vulnerability disclosure. We disclosed our findings to Google on 18 December 2012 under Google's *Vulnerability Reward Program* and Google provided (10 June 2013) the following response for publication.

> We thank the authors for sharing their research with us - Google encourages and supports security research in the community. Practically, an adversary with physical access to a victim's computer is very hard to stop, and this attack requires physical access as well as control over the network connection.
>
> Using a computer in a shared or public space (e.g., Internet kiosks) has a different threat and risk model compared to using your own computer. Most operating systems can be configured in a guest mode that protects or deletes the users' cookies and other profile information on [sign-out]. The guest mode in ChromeOS addresses this concern by design out of the box, or you can configure a ChromeOS device for Public Sessions (`http://support.google.com/chrome/a/bin/answer.py?hl=en&answer=3017014`). For other operating systems, there are several methods that can be used to address this risk. If you have no control over the computing environment, we recommend using a browsing mode that does not retain cookies (such as Chrome's Incognito mode or similar modes in other browsers).

We agree that it is difficult to defend against an adversary that has physical access to a victim's computer, nonetheless, we consider this difficulty a challenge rather than an insurmountable problem. Moreover, we have shown that Google could defend against the attacks we highlight, under the assumption that the victim's computer cannot be compromised by the adversary. Furthermore, we believe that web applications should be secure against adversaries that control the network connection, as per our discussion in Section 1.3. Google have confirmed (16 June 2013) that no financial reward will be issued on the basis that:

> when this paper was presented to us, there were already efforts underway that would stop this type of attack. As we can only reward issues that we were previously unaware of, this report was ineligible for reward.

Google have not clarified precisely which issues they were previously aware of nor what efforts are underway to stop this type of attack (other than the solutions included in their above response). At the time of publication, Google have not fixed the vulnerability.

4.3 Discussion: TLS termination modes

The absence of a distinction between TLS termination modes is crucial for our attack, in particular, we rely on the browser ignoring the fatal connection closure which is generated by dropping request 3 and our attack would be thwarted if the browser refused to execute the `onload` event upon such errors. Nevertheless, we believe that the vulnerability can be eliminated by modifying the application logic, as we shall discuss in the next section.

4.4 Countermeasures

As per Section 3.3, we recommend that all authentication is managed centrally or the sign-out procedure is handled by chaining a sequence of HTTP redirects over TLS. In addition, as a weaker, short-term solution, which is faster to deploy, we observe that adding an `onerror` Javascript handler to the images included in HTML pages could mitigate the attack: if the image fails to load, then an error message can be displayed to the user and the sign-out procedure can be interrupted.

5 Conclusion

We show how an adversary can exploit flaws in the sign-out procedures used by Helios, Google and Microsoft Live to prevent active sessions from terminating, whilst giving the user feedback that her sign-out request succeeded. Moreover, if the adversary can access the user's computer *after* the user believes she signed out, then we show that the adversary can cast votes on behalf of honest voters, take full control of Microsoft Live accounts, and gain temporary access to Google accounts. We stress that the attack does not rely on compromising the victim's computer (e.g., by installing malware). Consequently, shared computers – even uncompromised computers – cannot guarantee secure access to Helios, Microsoft (including Account, Hotmail, and MSN), nor Google (including Gmail, YouTube, and Search). In addition, we observe that Google and Microsoft do not protect all requests with HTTPS during the sign-out procedure and thus trivial attacks are possible; these trivial attacks can be prevented by using HTTPS, rather than HTTP, but this fix does not prevent our main attack.

Our attack vector is quite simple, but, nevertheless, the underlying vulnerability is present in the three systems that we have studied and has not been publicly disclosed. We believe this is due to a poor understanding of the security guarantees that can be derived from TLS and the absence of robust web application design guidelines. In publishing our results, we hope to raise awareness of these issues before more advanced exploits, based upon our attack vector, are developed.

Conceptually, our attacks are due to a desynchronization between the user's and server's perspective of the application state: the user receives feedback that her sign-out request has been successfully executed, whereas, the server is unaware of the user's request. It follows intuitively that our attack vector could be exploited in other client-server state transitions. Nonetheless, we believe that attacks can be avoided by reliably notifying the user of server-side state changes. Unfortunately, the HTTP protocol is unsuited to this kind of notification and we advocate the inclusion of such notification mechanisms in future standards, e.g., SPDY. Alternatively, notification mechanisms could be developed at the application level using technologies such as AJAX.

Acknowledgements

We are particularly grateful to Chetan Bansal, David Bernhard, Karthikeyan Bhargavan and Antoine Delignat-Lavaud for their suggestions.

References

[1] Ben Adida. Helios deployed at Princeton. `http://heliosvoting.org/2009/10/13/helios-deployed-at-princeton/`, 2009.

[2] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *EVT/WOTE'09: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. USENIX, 2009.

[3] Josh Benaloh, Serge Vaudenay, and Jean-Jacques Quisquater. Final Report of IACR Electronic Voting Committee. International Association for Cryptologic Research. `http://www.iacr.org/elections/eVoting/finalReportHelios_2010-09-27.html`, Sept 2010.

[4] Diana Berbecaru and Antonio Lioy. On the Robustness of Applications Based on the SSL and TLS Security Protocols. In *Public Key Infrastructure*, volume 4582, pages 248–264. 2007.

[5] Antoine Delignat-Lavaud, Alfredo Pironti, and Ben Smyth. Private communication, 29th April 2013.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, 2008.

[7] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory*, 29:198–208, 1983.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, 1999.

[9] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, Internet Engineering Task Force, 2011.

[10] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[11] Stuart Haber, Josh Benaloh, and Shai Halevi. The Helios e-Voting Demo for the IACR. International Association for Cryptologic Research. `http://www.iacr.org/elections/eVoting/heliosDemo.pdf`, May 2010.

[12] IACR Elections. `http://www.iacr.org/elections/`, 2013.

[13] Andrew J. Ko and Xing Zhang. Feedlack detects missing feedback in web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2177–2186, New York, NY, USA, 2011. ACM.

[14] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, pages 249–256. ACM, 1990.

[15] Donald Norman. *The design of everyday things*. Basic Books, 1998.

[16] Helios Princeton Elections. `https://princeton.heliosvoting.org/`, 2012.

[17] E. Rescorla. HTTP Over TLS. RFC 2818, Internet Engineering Task Force, 2000.

[18] Bruce Schneier. Government Secrets and the Need for Whistle-blowers. `http://www.schneier.com/blog/archives/2013/06/government_secr.html`, 2013.

[19] Bruce Schneier. IT for Oppression. *IEEE Security & Privacy*, 11(2):96, 2013.

[20] Ben Smyth and Alfredo Pironti. Attacking Helios: An authentication bug. YouTube video, linked from `http://www.bensmyth.com/publications/2013-truncation-attacks-to-violate-beliefs/`, 2012.

[21] Ben Smyth and Alfredo Pironti. Truncating TLS connections to access GMail accounts. YouTube video, linked from `http://www.bensmyth.com/publications/2013-truncation-attacks-to-violate-beliefs/`, 2012.

[22] Ben Smyth and Alfredo Pironti. Truncating TLS connections to access Hotmail accounts. YouTube video, linked from `http://www.bensmyth.com/publications/2013-truncation-attacks-to-violate-beliefs/`, 2013.

[23] C.A. Vlsaggio and L.C. Blasio. Session management vulnerabilities in today's web. *Security Privacy, IEEE*, 8(5):48–56, 2010.