**SAFEGUARD YOUR USERS PRIVACY**

Ensuring privacy is not as strait forward as you might hope. **Tim Badaczewski** explains and lists some best practices you can adopt to protect your users from privacy exploits.

Privacy concerns are an important emerging issue, and you see them in the news on a weekly basis these days. Many of us are used to remediating accessibility and security issues, but not enough web sites address holes in their privacy walls. There are simple solutions you can implement today to protect your applications in the future.

**PERCEPTION AND CONTEXT**

Content that must remain private is called "Personally identifiable information" (PII). This is any information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context. The determination of weaknesses in a privacy wall comes with a bit of fuzzy logic. We can’t control the security of external networks or the devices users’ access out apps with, yet we are responsible for any combination of factors that might compromise our apps.

In the last year government, social and ad driven websites have all been under a microscope as the perception of what is “private enough” has evolved. Part of that evolution is just the natural result of our acceptance of more integrated services in our life. Users enjoy the convenience and transparency of these integrations, but also assume the personal information they reveal won’t be exploited. As app creators we must recognize this and address it the same way we address efficiency, usability, accessibility and security.

**THE "ARE YOU SERIOUS?" SCENERIO**

Cathy sits in a busy coffee shop browsing the web on the shop’s free Wi-Fi as she sips her tasty beverage. One of her friends was just diagnosed with depression and she promised to help find them a support group. After a few searches she finds a group that’s local and a good fit for her friend, and uses the website’s contact form to see when they meet.

At the same time an attacker is monitoring all traffic on the coffee shop’s unsecured network and notices the traffic to the unencrypted domain. They can see the user-agents of the requests and quickly identifies Cathy. Viewing the unencrypted traffic the attacker now knows Cathy’s full name, the first name of her friend via the contact form she just filled out. Cathy’s social network accounts are searchable and after a few minutes of sleuthing the attacker finds the full name of her friend via an Instagram comment she made on one of Cathy’s photos. With the combination of information gained, the attacker now has enough information to extort her friend.

Your first thoughts might be, “This is farfetched! ... I have no control over the security of the coffee shops network! … My application can’t control what information social networks share!” This is all true, but the fuzzy aspect of privacy is the context. Looked at individually the different bits of information might not be considered personal, but the combination of that information is. Your application facilitates that context of information and it is up to you to break it!

**GET THE "CORRECT" TYPE OF CERTIFICATE**

The first and most effective line of defense in your privacy wall should be an SSL certificate. A decade ago encryption added significant overhead, but that’s not true today. SSL/TLS accounts for less than 1% of the CPU load, less than 10 KB of memory per connection and less than 2% of network overhead (*istlsfastyet.com*). Any noticeable overheard pretty much eliminated, so no excuses.

Unfortunately all SSL/TLS certificates are not created equal and you want to pay attention to a few bullet points to make sure you get the right one:

* **Certificate** – You should have a certificate signed using **SHA-2**, the industry is moving away from SHA-1 because of its weaknesses. Support for SHA-1 is slated to end in late 2016.
* **Protocol Support** – You want to makes sure you support **TLS 1.2**, that’s the most recent version of the protocol. Older versions are susceptible to browser exploits such as BEAST and POODLE.
* **Key Exchange** – The stronger the key used in the session handshake the less likely someone can gain access to the communication channel. You want a key **2048 bits or above**.
* **Cipher Strength** – You want **256 bit ciphers or above**, but this is a little tricky because it depends on what ciphers are available on your server. You also want to make sure you choose cipher suites that support your target clients.

**PERFECT FORWARD SECRECY (PFS)**

This feature ensures that the two keys used for the secure session handshake can’t be compromised. If an attacker can determine the key, than they can decrypt your encrypted data. Make sure you have TLS 1.2 enabled on your server and use a compatible list of ciphers suites. You want to make sure that your cipher suites support the clients your application targets. Clients are basically the browser version and operating system versions they run on (i.e. – IE 11 /Win 7, Firefox 35 / OS X or in the case of mobile Android 4.11).

Two good resources for generating cipher suites are:

* Apache and Nginx (*mozilla.github.io/server-side-tls/ssl-config-generator*)
* IIS (*nartac.com/Products/IISCrypto*)

**ENFORCE THE HTTPS PROTOCOL**

Just offering a secure connection is not enough, and just using a secure connection for portions of your website isn’t either. If you have an SSL than institute it site wide!

**Enforce server-side via Rewrite**

Forcing the protocol at the earliest point of communication is best, which usually means a header rewrite performed at the moment the server detects the request.

Apache (*via .****htaccess***)

RewriteEngine On  
RewriteCond %{HTTPS} off  
RewriteRule (.\*) https://%{HTTP\_HOST}%{REQUEST\_URI}

IIS (via **web.config**)

<rule name="HTTP to HTTPS redirect" stopProcessing="true">  
<match url="(.\*)" />  
<conditions>  
<add input="{HTTPS}" pattern="off" ignoreCase="true" />  
</conditions>  
<action type="Redirect" redirectType="Found" url="https://{HTTP\_HOST}/{R:1}" />  
</rule>

**Enforce client-side via HTTP Strict Transport Security (HSTS)**

A fairly new opt-in security feature enhances the enforcement of the secure protocol by registering it within the code of the browser itself. One registered, all future requests will be secure, even before the request to the target application is made!

Apache (*via .****htaccess***)

Header always set Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

IIS (via **web.config**)  
  
<httpProtocol>  
<customHeaders>  
<add name="Strict-Transport-Security" value="max-age=31536000; includeSubDomains; preload "/>  
</customHeaders>  
</httpProtocol>

**MINIMIZE EXTERNAL CLIENT-SIDE SCRIPTS**

External JavaScript references are given kudos for reducing domain requests as well as being more likely cached in the user’s browser. The problem with external references is that they are completely out of our control, and we have to take the good with the bad.

The popular share plugin AddThis was sighted for canvas fingerprinting in 2014. Fingerprinting involves using the HTML5 canvas to create hidden tokens that can be shared among advertising partners. In 2015 Facebook’s Graph API was found to be vulnerable to malicious apps gaining access to users private photos. Add most recently GitHub came under a DDoS attack facilitated by unencrypted sites referencing external ad network scripts that were modified to flood GitHub with zombie requests.

Let’s mitigate the risk by asking a few questions:

* Is there a server-side alternative that would work?
* Can I host the script within my application?
* Can this script be referenced via HTTPS?
* Does a smaller more secure alternative exist that fits my needs?

It’s this developer’s opinion that all external references should be secure, and if there is not secure option then they shouldn’t be referenced. It

**HARDEN GOOGLE ANALYTICS**

This is a small quick change, but since just about everyone uses this form of analytics tracking it’s an important one. If you want to make sure your users stay secure and anonymous add these two lines of code beneath the primary good snippet.

* **Anonymize IP** - When present, the IP address of the sender will be anonymized.  
    
  ga('set', 'anonymizeIp', true);
* **Force SSL** - By default, tracking beacons sent from https pages will be sent using https while beacons sent from http pages will be sent using http. Setting forceSSL to true will force http pages to also send all beacons using https.  
    
  ga('set', 'forceSSL', true);

**TOP TIP**

Keep any identifiable information out of the URL. It’s true that network sniffers can’t see it, but they’re still visible in web server logs. GET/POST all variables as part of the encrypted header.

**FACT**

We call the certificates that digitally bind cryptographic keys “SSL certificates”, but in reality the SSL protocol is considered insecure. Certificates issued today use the TLS protocol, TLS 1.2 being the most secure. This is why you will see SSL paired with TLS in many references.

**RESOURCE**

If you have an SSL and want to get a free scan to see if your certificate and server configuration is secured properly, visit: *ssllabs.com/ssltest.*