(P)WP Security

LEONARD ROSENTHOL
ADOBE SYSTEMS

Secure Contexts (WP)

- https://w3c.github.io/webappsec-secure-contexts/
  - This specification defines "secure contexts", thereby allowing user agent implementers and specification authors to enable certain features only when certain minimum standards of authentication and confidentiality are met.
- A number of OWP features require the use of a secure context
  - Service Workers
  - Access to device features (g. geolocation, microphone, camera)
    - Should a (P)WP have access to device features (eg. microphone, camera, location, etc.)?
      - Is it up to the author or publisher? To the UA? To the user?
  - Notifications
- On the web, this would be an https connection. Off the web, it's up to the UA.
- NOTE: You can't load insecure contexts from a secure context
  - Eg. you can't load http://louvre.fr/monalisa.jpg from https://mysite.com/book.html

Restricting Content (WP)

- **Using CSP (Content Security Policy - https://www.w3.org/TR/CSP/)**
  - A mechanism by which web developers can control the resources which a particular page can fetch or execute, as well as a number of security-relevant policy decisions.
- **Restricted CSS**
  - Solutions such as AMP (https://www.ampproject.org/) restrict the functionality for security reasons
- **Restricted SVG and MathML**
  - XML-based grammars have their own set of concerns
  - Plus ability to incorporate scripts and CSS directly
- **No Plugins**
  - Plugins still work in some browser engines (eg. CEF), but should that content be allowed?
  - Plugins introduce additional security risks but for what gain?
- **No Embed/Object**
  - What would a recursive publication look like?
- **Preventing “surprise”**
  - Access to some OWP features that should probably require a user consent: Geolocation, camera, microphone, desktop-notifications, push, full-screen, turning off mouse-cursor, system exclusive midi, clipboard-access …

Restricting JavaScript (WP)

- Certainly a lot of malicious content and exploits can be prevented if JS execution is disallowed by default, or even not permitted in the file itself.
  - NOTE: Several script-less hacks are equally possible (http://lcamtuf.coredump.cx/postxss/)
- However, JavaScript execution is fundamental to the OWP. Disallowing JavaScript would considerably cut down the power of WP.

- AMP also restricts some JS features for similar reasons
- Also what to do you with methods that block operations (eg. alert() or prompt())?

PWP Risks

- Do malicious (P)WPs with network access carry more risk than a malicious URL? **YES!**

- **User Expectation**: Users understand that accessing a web site involves network access, but users won't expect that opening a document can trigger the same activity.
- **Lack of Provenance**: When viewing a website, the user sees the service providing the content, but there's no such visibility for a PWP that might have made its way to a user through any path – Facebook, Dropbox, corporate archive, email.
- **Protection provided in browsers is based on domain/origin**: Services such as Google Safe Browsing (https://developers.google.com/safe-browsing/) allow browsers to identify malware sites (based on URL blacklists, domain reputation etc.) and warn the user.
- **Protection provided in browsers is based on content being available online***:* E.g., Googlebots crawl the web to identify badness. If that badness is packaged in a file (and sent via emails), it's harder for online crawlers to identify them.
- **Lack of Mechanisms for Controlling Malicious Content**: If there is a malware attack online, that can be fixed and will only impact the small number of users that would have visited the site during the attack. But for a PWP that contains such malware, it can linger and continue re-distribution. It's also not clear if AV scanning can help.
- **Lack of update mechanisms to fix vulnerabilities***:* It's difficult/impossible to update a PWP to fix a security vulnerability.

Origins (PWP)

- Each PWP needs to be considered as it's own "site", and cannot talk directly to any other PWP
  - EPUB 3.1: *Reading Systems need to behave as if a unique domain were allocated to each Content Document*
  - Similar to opening the OWP content in an incognito/in-private browsing session and then closing the session.
  - NOTE 1: This prevents local persistence across document open/reads
    - (https://w3c.github.io/webappsec-clear-site-data/#goals)
  - NOTE 2 : Indirect communication through a site and REST APIs are fine (as it aligns with the web model)

- How to associate an origin with a PWP? Does the Web Package solution help?
  - Can it scale from individuals to publishers?
  - And if we have an actual origin, how to then consider each PWP a "sub-domain" for that origin?
  - How to make this work inside a browser environment (eg. an online bookstore)?
  - TLS certs aren't the same as file/doc signing certs.

- Sub-Origins (https://w3c.github.io/webappsec-suborigins/)
  - This specification defines a mechanism for programmatically defining origins to isolate different applications running in the same physical origin.

Restricting Network Access (PWP)

- In the case of PWP's, should we restrict network access?
  - All
  - All but certain specific use cases (eg. fonts and streaming media)
  - Only when a PWP says that it wants it
  - Only when we trust it
- EPUB 3.1: *Reading Systems that enable scripting and network access also need to consider including methods to notify the user that network activity is occurring and/or that allow them to disable it.*
- Other considerations around Network Isolation
  - Blocking downloads
  - Blocking top-level navigations
  - Whitelisting of URL schemes that can be launched

Security issues due to non-updatability (PWP)

- If we encourage people to embed Javascripts in their documents, we may create a situation in which billions of copies of that JS are distributed without any mechanism for ever updating them.
  - Does distributing code on a massive scale without any updating mechanism seems wrong?
- Imagine some popular JS library has a serious flaw that isn't caught for a long time. Long enough for many, many, copies of the buggy JS to be replicated in documents. Some errors might be:
  - a calendar-related flaw which only surfaces after a particular date is passed;
  - inadvertent reliance on a bug in the JS engine, one which is subsequently corrected, breaking backward-compatibility with code that relied on it;
  - deliberate reliance on a third-party site (let's say, for analytics), but the third party goes out of business and the domain is picked up by someone else
- **There isn’t a viable solution, because you cannot update the library behind the original author’s back as you don’t know what that will do.**
  - E.g., they could be using a method that has become deprecated in newer versions. Or the behavior of a given method changes.

A little bit on Privacy: Phoning Home

- User Behavior Tracking
  - Primarily, “phone-home” checks prevent silent tracking of user actions on the document (e.g., knowing when a document was opened, using IP-to-geolocation where it was opened, knowing what the user read and for how long …).
  - Phone-home checks were implemented by in Acrobat 7.0.5 (2004) to solve this issue:
    - “Phone home notification [means] that when a PDF document attempts to contact an external server for any reason, the end user will be notified via a dialog box that the author of the file is auditing usage of the file, and be offered the option of continuing.”
- Spammer Abuse
  - Email clients *do not automatically download* external images that are linked into an HTML email (https://litmus.com/blog/the-ultimate-guide-to-email-image-blocking). One big reason for this behavior is to defeat a spammer technique wherein a spammer sends emails in bulk to thousands of email addresses, but with each email containing a unique image link.
- CSRF (https://www.owasp.org/index.php/Cross-Site_Request_Forgery)
  - If a user visits a malicious link on the Internet, that malicious webpage can mount various kinds of web attacks like CSRF that rely on the user’s ambient credentials (like cookies or HTTP authentication) being sent along with the request.

Questions