Cruel Intentions: A Security Analysis of Web Intents

Jenna Kallaher, Amal Krishnan, Paul Makowski, Eric Chen, Collin Jackson
Carnegie Mellon University
{*jkallahe, achemman, pmakowsk*}*andrew.cmu.edu*, {*eric.chen, collin.jackson*}*@sv.cmu.edu*

***Abstract*—Web Intents are a new web collaboration framework intended to bring the benefits of Android's Intent model to the web. Web Intents are currently implemented as a JavaScript shim, supporting several major localStorage-enabled browsers. A prototype native implementation is under development for Google Chrome. While Android's Intent model has previously been the subject of significant security review, Web Intents has not yet received similar scrutiny.**

In this paper, we present several attacks on the prototype implementations of Web Intents. We have communicated our recommendations on mitigating these attacks to the Web Intents developers. Our concerns have been acknowledged by the developers and several of our suggestions were adopted.

I. Introduction

The World Wide Web offers a plethora of services to assist users with their daily tasks. Some of these tasks include photo editing, file sharing, or even adding events to the users' personal calendars. Most often, developers are burdened with the decision of selecting the appropriate subset of services to integrate with their web applications. Unfortunately, the current web framework makes it difficult for developers to anticipate new services, and to select APIs based on their users' personal preferences.

Web Intents were proposed to solve this by facilitating interactions between web applications and application service providers. Web Intents are a client side framework that enables resource sharing and communication between different web applications. To utilize this framework, the content providers must first assign each resource with an *intent action* attribute (e.g., share); the user proceeds by evoking a previously registered intent to perform the action specified by the action attribute.

During our security evaluation of Web Intents, we discovered four attacks: a privacy attack that can be used to track users across multiple sessions, a denial of service attack on intent storage, an attack that overwrites benign intents with malicious intents, and a potential venue for login CSRF attacks. We proposed defenses that either partially or fully mitigate the attacks we discovered. The defenses are as follows: first, we require users to explicitly grant permission when an intent is registered. Second, the browser should provide visual indicators differentiating intents that are registered over SSL. Last but not least, intents storage should he isolated based on the intent origin. The Web Intent working group responded positively to all of our concerns and are working to implement most of our defenses.

```
<intent action="share" type="image/*"
    href= "http://photoshare.tld/share"
    title="Image Sharing" />
```

Figure 1. An example of intent registration.

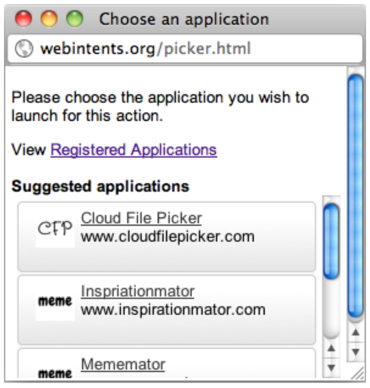
We provide background on the Web Intents framework in Section II. In Section III, we enumerate the discovered avenues of attack on Web Intents and their threat model. In response to these attacks, we discuss our recommended defenses in Section IV. In Section V, we provide a summary of developer response to our presented attacks and defenses. Finally, we survey related work in Section VI and conclude in Section VII.

II. Web Intents Overview

Web Intents simplify how websites are able to interact with other applications and services. Current web collaboration solutions impose a non-trivial cost increase for *each* supported API, whereas Web Intents allows developers to consolidate the cost increase into support for a *single* API. Subsequent addition and removal of supported services does not impact the Web Intents adoptee's cost.

Let `photoshare.tld` be a popular photo sharing website. Traditionally, web developers who want to allow content sharing with `photoshare.tld` would have to learn `photoshare.tld`'s API and would have to modify their website to support it. By employing Web Intents, the proprietors of `catpictures.tld` no longer need to use `photoshare.tld`'s API. Instead, `photoshare.tld` would register an intent with the user's browser indicating that it would like to handle image files with a `share` action. This intent will persist until the user directly removes the intent. The intent is registered by embedding an HTML tag similar to the one depicted in Figure 1. On a subsequent visit to `catpictures.tld`, the user clicks a "Share" button, prompting `catpictures.tld` to fire an intent for `image/*` files with an `action` attribute of `share` (refer to Figure 1). The browser will provide the user with a list of options for sharing the image, as seen in Figure 2. Since `photoshare.tld` previously registered an intent to handle such an `action`, its photo sharing service is included in this list of choices that is presented to the user.

Going back to Figure 1, the `action` field indicates the type of action to be taken. Currently, documented


Choose an application

webintents.org/picker.html

Please choose the application you wish to launch for this action.

View Registered Applications

Suggested applications

Cloud File Picker
www.cloudfilepicker.com

Inspriationmator
www.inspirationmator.com

Mememator

Figure 2. Intent selection interface for the JavaScript shim implementation of Web Intents

actions include `discover`, `view`, `pick`, `share`, `edit`, `subscribe`, and `save`, but developers are free to create their own actions simply by specifying a unique value for the `action` attribute of the registered intent.

Below, we expand on the uses for each intent attribute.

- **action**: Currently documented `actions` are URLs that double as reference points for developer documentation. The URL scheme is not required in this field. Global uniqueness, however, is required in order to maintain specificity between related actions.
- **type**: The `type` attribute indicates the data type that is supported in the context of the registered action.
- **href**: The `href` attribute is a URL that points to the location that handles the action.
- **title**: The `title` attribute stores a user-friendly name for the action, to be displayed to the user when this Intent is displayed in a dialog prompt.

Web Intents allows cross origin communication between two websites as long as both websites follow the Web Intents protocol. The intent-firing website (`catpictures.tld`) requires no knowledge of the implementation on the intent-registered website (`photoshare.tld`) because the Web Intents API is standardized across all intents. When a user chooses to share an image on a Web Intents-enabled website, an intent is fired and a list of all websites that have previously registered (`action`, `type`) tuples matching the fired (`action`, `type`) tuple will be presented to the user. In the prototype Chrome implementation, this intent list is a non-spoofable chrome element, populated by the browser with supporting intent handlers. The user then chooses which website (e.g. `photoshare.tld`) she wishes to use to share the image.

III. Attacks on Web Intents

In this section, we present our security evaluation of Web Intents. We have discovered four attacks on the most recent implementation of Web Intents. All of these attacks were verified by the Web Intents working group.

A. *Threat Model*

In our evaluation of Web Intents, we consider two types of attackers, the web attacker and the network attacker. The web attacker has control of a malicious website; she also has the ability to lure users into visiting her website. We do not assume users to fully trust the malicious website, that is, users may refuse to install intents provided by the web attacker. However, this assumption is moot because, prior to our research, intents could be installed automatically without users' consent. In addition to a standard web attacker, some of the attacks we discovered require additional network capabilities. Network attackers can read, modify and deny any network traffic, but do not have the ability to break existing encryption scheme or read encrypted traffic.

B. *Privacy Attacks*

A web attacker can track Web Intents users across multiple browsing sessions, even when browsing data is cleared and IP addresses change. To launch this attack, the attacker first creates intents with unique `href` attributes, then lures the user into visiting her page. This unique intent identifier can then be used to associate the user's activity with previous session. This attack is similar to the user tracking technique previously described by Samy Kamkar [1]. In the current prototype implementation, clearing browsing history is insufficient for clearing tracking indicators. Intents are stored permanently by the browser until the user explicitly removes the intent. Therefore, this type of user tracking technique has the potential to to record user activity for much longer periods of time than other techniques (e.g., CSS history sniffing [2]).

C. *Intent Registration Denial of Service*

In the current implementation of Web Intents, websites are not limited in the number of intents or the size of intents that can be registered. A web attacker can effectively cause a denial of service by filling the browser's Web Intents storage. Prior to our raising of this issue, the Web Intents working group has not considered limiting the space allocated to Web Intents. To solve this problem, we propose a similar storage architecture as the cookie storage, where intent registration is limited on a per-origin basis. Furthermore, we believe a user confirmation is needed when registering for an intent. Details of our proposal is described in Section IV-A.

D. *Persistent Effects via Unintentional Intent Registration*

An adversary can leverage an existing XSS vulnerability to overwrite a previously registered intent handler in the context of `victim.com`. Web Intents will allow the attacker to amplify her XSS capabilities by causing her malicious intent registration to remain even after the XSS vulnerability is fixed on `victim.com`. This is particularly interesting when exploiting a reflected XSS since Web Intents will allow the effects of the attack to persist. Once an intent is registered, the user's browser will store the intent indefinitely. This behavior escalates a simple reflective XSS capability to a semi-persistent intent corruption capability. Additionally, the user will see no indication that the intent has changed since the initial registration.

Adversaries with network capabilities can perform similar attacks by overwriting securely registered intents with unsecure intents. For instance, consider a registered intent whose `href` attribute points to a page served over SSL. An attacker could overwrite this intent such that the intent's `href` attribute points to a non-SSL page, paving the way for a mixed-content scenario that would otherwise not exist. Suppose `example.com` registers an intent for the purpose of sharing images. `example.com` might choose to create two intents, one for normal image sharing and one for secure image sharing. The network attacker can modify this registration attempt in order to gain access to an image that the user believes is being shared securely. During the initial serving of the intent, the network attacker could change the URL for the secure photo sharing page to the URL for the unsecure photo sharing page. This can also be done after the fact, by causing an intent overwrite. Even if the user is prompted for intent registration (for this overwrite), it is likely that the user will accept a registration attempt from a website that she has previously visited (e.g. `example.com`). When the user decides to share an image using the modified intent, the network attacker would be able to observe the photo. This attack allows the attacker to access data that the user believes was shared securely. To fix this vulnerability, browsers must either not allow intents to be registered through unencrypted pages, or forbid non encrypted pages from overwriting intents registered securely.

E. Login CSRF

User interface design problems may not directly send information to an attacker; however, the attacker may still be able to obtain information by sending it through another site. One of the problems with current intent registrations is that they do not offer the capability to tie the registration to a user's account. A user could have two accounts on a website that registers intents, but the intents were only registered while the user was logged into one of the accounts. If the user is logged into one account and attempts to execute an action, the action may be automatically taken using an unintended account. The current prototype implementation of Web Intents does not indicate which account will be used to carry out the action, nor lock registrations to the account that was active at the time of registration. Without the ability to tie a particular account to a registered intent, login CSRF [3] becomes very powerful in the context of Web Intents, since it allows a web attacker to trick the user into divulging information to an attacker-controlled account with minimal user interface indication. To see a real life example of this attack, consider a Flickr intent that can be used to post pictures onto the user's Flickr account. Furthermore, assume that Flickr has a login CSRF vulnerability (i.e., the attacker has the ability to silently log the user into the attacker's account in the background). The attack proceeds as follows:

1) User installs the benign Flickr intent onto her browser.
2) User visits the malicious website, which launches a login CSRF attack on Flickr in the background.
3) User visits `PrivatePhoto.com` and uses the intent registered in Step 1 to upload her private photo to her Flikr account.
4) Since the user is currently logged in as the attacker (due to the attack carried in Step 2), the attacker will gain access to the user's private photo.

In the above example, Web Intents displays only the services associated with an intent action and does not indicate the associated account. If the user selects a vulnerable service, intent data would be sent to the attacker account without the user's knowledge. This attack is further aggravated by the lack of integrity in HTTP cookies that makes login CSRF attacks difficult to completely defend against [4].

IV. Defenses

We propose several modifications to the design of Web Intents in order to mitigate the first three vulnerabilities we identified. The defenses proposed in this section have been communicated to the Web Intents developers, who have either integrated our proposals or are planning to do so soon. See Section V for more details.

A. User Confirmation for Registrations

One simple way to mitigate the threat posed by several of the presented attacks is to require user confirmation when a website wishes to register an intent. Such a feature would significantly restrict a web attacker's capabilities. By conducting intent validation such as prompting the user prior to intent registration, the web attacker's ability consume and free almost arbitrary amount of space in the Web Intents storage is severely restricted. Without the ability to consume and free this space, the web attacker cannot effectively conduct the previously described denial of services attack variants. Additionally, such user interaction would be more analogous to Android's Intent registration behavior. While Android users are not directly prompted for the registration of Android Intents, they are directly prompted for the installation of applications. By requiring user interaction to install and remove applications, and therefore those applications' registered Intents, Android effectively provides the user with a means to manage her Intent database. We envision that native implementations of Web Intents will provide the user with similar discretion. The Web Intents developers agree with this recommendation and have stated that future releases of the JavaScript shim and any native implementations will prompt users prior to registration.

B. Authenticity Indicators in User Interface

To mitigate network attacks, we propose that Web Intents should present visual cues to users that clearly differentiate between intents registered over HTTP and intents registered over HTTPS. Such cues would likely be reminiscent of the visual cues employed my all major browsers for HTTPS-secured sites. However, we acknowledge that it would be difficult to train users to make well-informed security decisions based on these indicators. A more conservative approach would be to require HTTPS URLs for all intent registrations.

C. Isolation of Securely Registered Intents

In addition to presenting visual indicators for secure intents, we believe securely registered intents should also be kept in a separate storage as intents registered via HTTP. More specifically, Web Intents should leverage the security boundaries called for by the same origin policy. This would guarantee two things. First, a network attacker would not be able to overwrite secure intents with insecure ones. Second, a network attacker would not be able to overflow the secure intents storage by registering a large number of insecure intents. (This distinction is unnecessary if HTTPS is required to register Web Intents.)

V. Developer Response

The Web Intents developers responded quickly to the issues raised in this paper [5]. A "security" bug tag was created almost immediately after we had contacted them and is being used to track development progress toward addressing the presented concerns [6].

In future native implementations, Web Intents will require user approval for intent registration. This behavior mitigates the web attacker's capability to cause a Web Intents denial of service. Additionally, the developers plan to only offer the JavaScript implementation over HTTPS, thereby alleviating the concerns raised around modification by a network attacker for privacy-violating and mixed-content outcomes advantageous to the network attacker.

VI. Related Work

While the security implications of Web Intents have not been previously subject to external evaluation, the design and goals behind Web Intents are derived from browser extension security and Android's Intent framework, both of which have undergone such an evaluation.

Several recent papers [7], [8] have focused on the security implications of Android Intents and how application developers must go about utilizing the Android Intent framework in order to safeguard the confidentiality and integrity of their applications' code and data. Due to the closely related nature of Android Intents and Web Intents, threats and attack vectors applicable to Android Intents are in some ways parallel and applicable to Web Intents. Android Intents are vulnerable to a wide variety of attacks. Most of these attacks occur because Intents are used for both intra and inter-application communication. We have found attacks on Web Intents that are similar to previously discovered attacks on Android Intents like Activity Hijacking and Service Hijacking [8]; the means of exploitation are, however, very different as one is an inter-web application communication mechanism while the other an IPC mechanism.

VII. Conclusion

Prior to this paper, Web Intents had not been subject to an external security evaluation. Our evaluation has uncovered a number of attacks against current implementations of Web Intents. As Web Intents is still very much a work in progress and has yet to be utilized by any significant number of websites, we believe the timing of our evaluation may provide the most benefit due the relative lack of compatibility issues that would arise from major Web Intents modifications.

We offer recommendations on defending against our attacks in future browser implementations of Web Intents. The vulnerabilities presented were acknowledged by the developers of Web Intents, who have agreed with our most of our recommendations and are tracking progress toward their resolution.

References

[1] "Evercookie," http://samy.pl/evercookie/.

[2] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson, "I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, 2011.

[3] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2007.

[4] A. Bortz, A. Barth, and A. Czeskis, "Origin Cookies: Session Integrity for Web Applications," in *Web 2.0 Security and Privacy (W2SP)*, 2011.

[5] "Security research," 2011, email exchange. https://groups.google.com/forum/?fromgroups#!topic/web-intents/vYbcZJ9aJrE.

[6] "Security issues." 2011, https://github.com/PaulKinlan/WebIntents/issues?labels=Security&sort=created&direction=desc&state=open&page=1.

[7] J. Burn, "Mobile Application Security on Android," in *Black Hat USA*, 2009.

[8] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000018