Exhibit K

Protecting Privacy with Referrers

By Matt Jones on Monday, May 24, 2010 at 8:24pm

Late last week, we quickly fixed an issue after being contacted by a *Wall Street Journal* reporter regarding an unintentional oversight in the data shared with our advertisers by your browser when you click some ads on Facebook. This occurred in the referrer link visible to advertisers when someone clicked on an ad.

A little background: In some cases the referrer could contain the user ID of a profile you visited, including your own, but we were not aware of any way that a user ID on the referrer could identify the person who clicked on the ad. We've been testing different solutions to remove user IDs completely from referrer URLs since their inclusion was first brought to our attention.

However, in a rarely occurring case, advertisers knowledgeable about the structure of Facebook's URLs could use the referrer to determine when someone who clicked on an ad had been viewing his or her own profile, thus potentially enabling them to infer the user ID of that person. We have no reason to believe that any advertisers were exploiting this, and doing so would have been a violation of our terms. To our knowledge, none did.

It's also important to point out that we don't share personal information with advertisers, and we never sell any of your information to anyone. The way our advertising system works is advertisers tell us what types of people they want to reach and we take their ad and serve it accordingly.

We want to do far more than the industry standard, though. Today, we began rolling out a change to completely remove all user IDs from appearing in referrer links before web browsers send the links to external websites, including to advertisers. We've been working for the past few months on this change. It is live for the Firefox, Chrome, Safari and Opera browsers, and we're working on a solution for Internet Explorer.

Below, we've also published technical details of how we made this change so that other websites can use the approach to remove user IDs from their referrer links if they choose.

We're confident that this new change, along with our earlier fix, goes beyond industry standards to further protect your privacy.

The technical implementation

Here at Facebook, we're all about understanding how people interact with our site – including how they end up here from across the vast expanse of the internet. We're not the only ones, though – most web sites want similar insights about the people who use them.

Despite its tragic misspelling, the HTTP standard's "referrer" header sent by browsers gives websites the information they need to see how users found them, and how they explore the sites once there.

Referrers: not always welcome

But sometimes referrers just don't belong – maybe there is sensitive information in a URL, or maybe a site just doesn't want its users' browsers telling others how they use the site. While most browsers give their users the option to disable this feature, not everyone does so, and there is no way for a web site to explicitly tell a browser not to send a referrer.

Facebook is one site where referrers don't really belong. As part of our continued efforts to protect users' privacy, we proactively protect our users from exposing how they navigated to an external site. To this end, we have designed a redirector with the following properties. The redirector must:

1. Successfully redirect in all cases

2. Remove the referrer of the page on which the user clicked
3. Still convey that a click happened on Facebook.com
4. Not be an open redirector

Implementing this functionality in a safe, cross-browser way is not a simple task, so here we discuss one method for accomplishing it.

Our friends, the web pages

For the purposes of this discussion, we'll refer to three web pages:

- A.com/source
- A.com/redirect
- B.com

Suppose the user is viewing content on A.com/source, and clicks a link that purports to go to B.com. A.com wishes to send the user to B.com in such a way that B.com knows A.com sent them the click - but without disclosing that the user clicked to their site from A.com/source specifically. So instead of sending the user directly to B.com, they can first send the user to A.com/redirect. This extra step also has some additional benefits (e.g. the ability to block the redirect if B.com is found to be malicious) – but we won't get into those here.

So how can we make this happen? There are number of approaches, each with its own advantages and disadvantages. You can see each approach outlined in the following table, and discussed in detail below it.

	Firefox	Webkit (Safari / Chrome)	IE6/7/8	Opera
302	source	source	source	source
Refresh	[blank]	redirect	[blank]	redirect
Meta Refresh	[blank]	redirect	[blank]	redirect
document.location.replace()	redirect	redirect	[blank]	redirect
anchor click()	fail	fail	redirect	redirect
shimmed anchor click()	fail	fail	redirect	fail

The discussed redirect methods and the referrers they send in each browser.

The 302

The simplest way to redirect is with an HTTP 302 Found response and an HTTP header "Location:http://B.com". However, in the majority of browsers, this type of redirect will show a referrer of "A.com/source" - which we don't want.

- PHP code:

```
header('Location:http://B.com');
```

The Refresh

The next simplest way to redirect is with an HTTP 200 response, and an HTTP Refresh header, "Refresh:0;URL=http://B.com". The behavior of this type of redirect varies across browsers.

- In Firefox, and Internet Explorer (IE), this type of redirect will result in a blank referrer being sent as the referrer to B.com. Good for privacy, but bad for B.com's ability to understand where its traffic comes from. We can do better.
- In Opera and Webkit browsers (Safari, Chrome, iPhone, and Android), this type of redirect will result in "a.com/redirect" being sent as the referrer to B.com - perfect!

- PHP code:

```
header('Refresh:0;URL=http://B.com');
```

The Meta Refresh

Similarly, we can send an HTTP 200 response, and some HTML with a meta tag. This is equivalent to sending the Refresh HTTP header.

- HTML code:

```
<html>
<head>
<meta http-equiv="refresh" content="0;URL=http://B.com" />
</head>
</html>
```

Document.location.replace()

Getting a little more tricky, assuming the user's browser supports JavaScript, we can send an HTTP 200 response and a snippet of JavaScript that says: "document.location.replace ('http://B.com');". This type of redirect also behaves differently across browsers.

- In Firefox, Webkit, and Opera, this type of redirect will result in "a.com/redirect" being sent as the referrer to B.com. Success!
- In IE, this type of redirect will result in a blank referrer being sent to B.com. Not ideal.
- HTML code:

```
<html>
<body>
<script>document.location.replace(”http://B.com”);</script>
</body>
</html>
```

Anchor click()

To find a viable solution on IE, we need to be a little bit more clever. If we again send an HTTP 200 code but some different JavaScript, we can make it work.
Specifically, we can send HTML containing an anchor () tag and some JavaScript that tells the browser to "click" it: "document.getElementById ('a').click();". One drawback to this approach is that it alters the behavior of the “back” button. When this approach is used, users who click “back” on B.com will go back to A.com/redirect, and then be immediately re-redirected to B.com.

- In Firefox and Webkit, this completely fails - click() does nothing.
- In Opera, IE6, IE7, and IE8, this sends a referrer of "a.com/redirect" to B.com. Success!
- HTML code:

```
<html>
<body onload="document.getElementById('a').click();">
<a id="a" href="http://B.com"></a>
</body>
</html>
```

2-way shimmed anchor click

We can get around the anchor click's annoying behavior by making a.com/redirect a little smarter. It turns out that IE persists some of the dom state in a.com/redirect even if you navigate away, then hit the back button and return to it. So we store a form with a hidden input that's initialized to 0, and every time a.com/redirect is hit, we increment a counter in that hidden input. If, when the page loads, the counter is 0, it anchor-click redirects to B.com. If the counter is even or odd (but >0), it uses window.history.go() to go forwards or backwards accordingly. (If the counter is odd, the user hit "back." If it's even, they hit "forward.") To the user, it's truly as if a.com/redirect doesn't exist - almost like the simpler methods in other browsers.

- Like the vanilla anchor.click(), this fails in Firefox and Webkit.
- In IE, this works as expected - and works around the break in the back button.
- In Opera, the first redirect works but unfortunately hitting the back button just leaves you on A.com/redirect.

- HTML code:

```
<html>
<head>
<script>
function onloadHandler() {
  if (document.refreshForm.visited.value == "") {
    document.refreshForm.visited.value = 1;
    document.getElementById("a").click();
  } else if (parseInt(document.refreshForm.visited.value) % 2 ==
1) {
    document.refreshForm.visited.value = parseInt
(document.refreshForm.visited.value) + 1;
    window.history.go(-1);
  } else {
    document.refreshForm.visited.value = parseInt
(document.refreshForm.visited.value) + 1;
    window.history.go(1);
  }
}
</script>
</head>
<body onload="onloadHandler();">
<form name="refreshForm">
<input type="hidden" name="visited" value="" />
</form>
<a id="a" href="http://B.com"></a>
</body>
</html>
```

So we've found redirect methods that work in all the major browsers, even if none of them works in every browser. So all we have to do is look at the user agent when we're rendering A.com/redirect, choose the right method, and send users on their merry way.

Keeping it safe

We also need to make sure that requirement (4) is fulfilled – we need to not be an open redirector. That is to say that A.com/redirect needs to only work for the person it's intended for, at the time they clicked it. To that end, we include a parameter that must be correct for the redirect to go through. This parameter of course does not convey any identifiable information about the user.

A backup plan

There's one caveat, though - what if some users' browsers don't support JavaScript, they have a plugin that prevents it from executing, or these methods otherwise fail? We still want them to end up on B.com - after all, they clicked on its link! So along with all of these methods, we also send a "Refresh:1;URL=http://B.com" header, which tells the browser to redirect to B.com after 1 second. This way if for some reason the JavaScript redirects fail, users will still end up at their destination and everybody wins. The only drawback to this method is that some browsers send a blank referrer when it's used, so affected users will send a blank referer. Since this is only a small percentage of users, we can accept this tradeoff.

We've implemented these methods to prevent user IDs and usernames from appearing in referrers sent by browsers when our users visit other sites from Facebook. We hope that other websites with similar needs can use these methods too. We would like support in web browsers for safe redirects, but until then, we're looking for any feedback or ideas for other approaches. Please share them in comments here.

Matt Jones, an engineer on Facebook's Site Integrity team, likes to convince things to work in ways they weren't designed to.