MBSD Technical Whitepaper

A few RPO exploitation techniques

Takeshi Terada / Mitsui Bussan Secure Directions, Inc.

June 2015

Table of Contents

1. Introduction

RPO (Relative Path Overwrite) is an elaborate attack technique publicized by Gareth Heyes in 2014 [1]. In essence, this attack utilizes a crafted URL (typically with a PATH_INFO), to force the target Web page to load itself as a stylesheet, when it contains both path-relative stylesheets and attacker-controllable contents.

In June 2015, MBSD conducted a research on this topic and discovered some new attack techniques. In this paper, we first describe path manipulation techniques specific to some client / server environments in the next section (§2). Then, some miscellaneous technical topics are described; a technique to forcefully enable IE's CSS expression using CV (§3), attack possibility on non-stylesheet relative URLs (§4), and a related vulnerability discovered in CakePHP framework (§5). In the next section, countermeasures are described.

Note that this paper is not an extensive or detailed guide of RPO, but is focusing on new techniques on it. More extensive and detailed information on RPO can be found in the original blog post [1] and PortSwigger's blog [2].

2. Path manipulation techniques

The original RPO attack has a few (implicit) restrictions:

(1) A vulnerable HTML and the stylesheet loaded by the HTML must be the same file.

(2) An attacker cannot manipulate query string parameters of the loaded stylesheet URL.

In this section, some attack techniques to circumvent these in certain environments are explained. The basic method is tricking browsers or servers by utilizing differences in how Web servers or browsers interpret URL paths.

For instance, characters such as a dot (`.`), slash (`/`), backslash (`\`), question mark (`?`) and semi-colon (`;`), and their encoded equivalents have special meanings in URL. Servers and browsers may interpret these differently. The differences in interpretation are what attackers can utilize to extend attack possibility of RPO.

2.1. Loading another file on IIS/ASP.NET

Before proceeding to our own findings, we would like to introduce a technique discovered by Soroush Dalili [3] (and later re-discovered by us) as a good example of such techniques.

Dalili showed that it is possible to trick browsers into loading another file as a stylesheet on IIS/ASP.NET, by using a URL containing `%2F` (or `%5C`) to traverse the path.

Dalili's PoC URL is the following.

```
http://sdl.me/demo/RPO/anotherpage.css.aspx/%2f..%2f..%2f..%2fdemo/RPO/test.aspx
```

The URL returns the content of `"/demo/RPO/test.aspx"`, because `%2F` and slash are treated equally on IIS/ASP.NET.

```
<link href="../../style.css" rel="stylesheet" type="text/css" />
↓
Doc base: /demo/RPO/anotherpage.css.aspx/%2f..%2f..%2f..%2fdemo/RPO/
CSS path: /demo/RPO/anotherpage.css.aspx/%2f..%2f..%2f..%2fdemo/RPO/../../style.css
          /demo/RPO/anotherpage.css.aspx/style.css
```

Meanwhile, the link element in the HTML (shown above) loads a stylesheet from `"/demo/RPO/anotherpage.css.aspx/style.css"`, as browsers do not treat `%2F` as a path separator. The PATH_INFO part of the stylesheet URL, which is `"/style.css"` in this example, is just ignored on the server side.

That means the first of the two restrictions mentioned above, which is regarding file paths, is successfully circumvented. Note that this sort of technique is quite convenient for attackers, because the URL of an attacker-controllable content is sometimes different from that of the Web page containing path-relative stylesheets.

2.2. Loading another file on Safari/Firefox

What is unique to Safari regarding path interpretation is that it does not decode encoded dots (and other characters) in URLs. Attackers can use this for another-file-loading attacks.

Suppose a Safari user accesses the URL below.

```
http://host/member/profile_photo.php/.%2E/top.php
```

Safari sends the URL path to the server (PHP/Apache here) exactly as it is. The server resolves `".%2E/"` in the path and returns the content of `"/member/top.php"`.

Suppose the response contains the following script element.

```
<script src="../js/jquery.js" type="text/javascript"></script>
↓
Doc base: /member/profile_photo.php/.%2E/
JS path:  /member/profile_photo.php/.%2E/../js/jquery.js
          /member/profile_photo.php/js/jquery.js
```

As Safari does not decode encoded dots in URL, the JS path loaded by the element will be `"/member/profile_photo.php/js/jquery.js"`. This means a file, different from the main HTML, is successfully loaded by the attack. In this case, the image returned by `profile_photo.php` is executed as JavaScript on the browser.

As for Firefox, it is unique only in how it handles trailing encoded double dots.

Suppose a Firefox user accesses the URL below.

```
http://host/member/profile_photo.php/.%2E
```

Firefox sends the path to a server without resolving the last encoded dots. The server resolves it, and then returns the content of `"/member/"`.

Suppose the content contains a script element below.

```
<script src="js/jquery.js" type="text/javascript"></script>
↓
Doc base: /member/profile_photo.php/
JS path:  /member/profile_photo.php/js/jquery.js
```


As Firefox leaves trailing dots unresolved when determining the JS path, the JS path loaded by the element will be `"/member/profile_photo.php/js/jquery.js"`, which returns the content of `profile_photo.php`.

Note that, in either attack, attackers cannot control the loaded resource path as freely as they can on ASP.NET. Specifically, the former attack requires the loaded resource path begin with `"../"`, and the latter one requires the HTML path end with a slash.

2.3. Loading another file on WebLogic/IE

Like IIS/ASP.NET, WebLogic treats `%2F` and slash equally, but a aforementioned attack on ASP.NET is not simply applicable to WebLogic, because of the two reasons: (1) WebLogic needs semi-colon to add a string to the end of a URL, and (2) a string like `"..%2F"` comes after semi-colon does not cause traversing on the server (WebLogic completely ignores a string after the first semi-colon occurrence).

Therefore another approach like the one shown below is needed for a successful attack.

```
http://host/aaa/Test1Servlet;/0/.%2E/.%2E/Test2Servlet;/0/
```

This URL returns the content of `"/aaa/Test1Servlet"`, because, as mentioned above, WebLogic simply ignores a string after the first semi-colon. Suppose the content contains the link element below.

```
<link href="../style.css" rel="stylesheet" type="text/css" />
↓
Doc base: /aaa/Test2Servlet;/0/
CSS path: /aaa/Test2Servlet;/0/../style.css
          /aaa/Test2Servlet;/style.css
```

`".%2E/"` in the original URL is decoded to `"../"` when browser determines the stylesheet URL, thus the resulting stylesheet's path sent to the server will be `"/aaa/Test2Servlet;/style.css"`, which serves the content of `"/aaa/Test2Servlet"`. This means an HTML file different from the first HTML is successfully loaded as a stylesheet.

The attack requires both (1) `".%2E"` occurrences in the initial HTTP request are sent as it is, and (2) these are decoded in the next stylesheet loading request. Both of these are only met on IE, when the attack URL is provided in a Location header.

```
HTTP/1.1 302 Moved Temporarily
Location: http://host/aaa/Test1Servlet;/0/.%2E/.%2E/Test2Servlet;/0/
```


In other words, this technique does not function, if the attack URL is provided in `<a href="...">` or just typed in the address bar. Because, in these cases, IE immediately resolves `".%2E/"` in the path before sending the initial request to the server.

2.4. Loading file with query string on WebLogic+Apache

A system architecture that places Apache in front of WebLogic is often seen in Java-based large corporate systems. For this constitution, Oracle provides an Apache module (mod_wl.so), which behaves like a reverse proxy between them.

Interestingly, the module normalizes URL paths in a quite unique manner, before passing the path to back-end WebLogic. Specifically, it decodes `%3F` in the URL path into `"?"`, which can obviously confuse URL parsers.

```
http://host/aaa/Test1Servlet%3Fparam={}*{color:red}/0/
```

Again, suppose the content contains the link element below.

```
<link href="../style.css" rel="stylesheet" type="text/css" />
↓
Doc base: /aaa/Test2Servlet%3Fparam={}*{color:red}/0/
CSS path: /aaa/Test2Servlet%3Fparam={}*{color:red}/0/../style.css
          /aaa/Test2Servlet%3Fparam={}*{color:red}/style.css
```

In this case, the final stylesheet's path being sent from browser to Apache is `"/aaa/Test1Servlet%3Fparam={}*{color:red}/style.css"`. This is because browsers regard `%3F` as a part of URL path, thus browsers do not remove the part when determining final stylesheet's path.

As already mentioned, `%3F` in the stylesheet's URL is decoded by the module, and then sent to back-end WebLogic, so that the part after `%3F` is treated as a query string in WebLogic. That means the second one of the two restrictions shown at the beginning of this section, which is regarding query strings, is successfully circumvented.

This behavior of the module is convenient for attackers when a target servlet reflects a query string parameter (`"param"` in the above example).

2.5. Attack possibility in other environments

As shown in techniques for ASP.NET and WebLogic, `%2F` in URL path can be a convenient tool for another-file-loading technique. In recent versions of Apache 2.2, however, any

occurrence of %2F in URL path unconditionally triggers 404 (not found) errors by default.

Still, a directive "AllowEncodedSlashes On" in its configuration can change the behavior [4]. An attack, similar to that on ASP.NET using %2F to load another CSS file, is possible if the directive is set to the on state.

We are not sure how common this configuration is in Apache installations, but a few high-profile Web sites including Facebook are confirmed to treat %2F and slash equally; it does not immediately mean these sites are vulnerable to RPO though. In general, RESTful Web sites, which normally accept various inputs in URL paths, are likely to enable the directive.

Similarly, recent versions of Tomcat have configuration option "ALLOW_ENCODED_SLASH", which is the equivalent of Apache's "AllowEncodedSlashes". Like Apache, this Tomcat's option is off by default. In addition, unlike WebLogic, normal Servlets and JSPs on tomcat do treat a slash occurrence after a semi-colon as a directory separator. Therefore RPO attacks on Tomcat are unlikely to succeed.

There are other environments that are not covered in our research. As a hint for a further research, a few more ambiguous path examples are listed below.

```
Contiguous slashes       /aaa/bbb//.%2E/ccc/
Semi-colon               /aaa/bbb/;/.%2E/ccc/
Backslash                /aaa/bbb\./.%2E/ccc/
```

Besides, browsers' behavior of truncating overlong URLs was tested. In these tests, we could not find out effective usage of them, but they can possibly help some researchers and pentesters in certain situations.

3. Forcing IE's CSS expression via CV

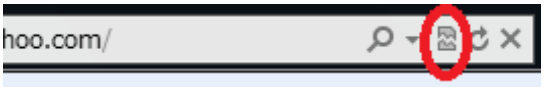
In RPO attacks that overwrite stylesheet paths, there are various attack techniques that do not require IE's CSS expression to run. Such techniques are well covered in PortSwigger's blog [2]. However, CSS expression is still a wanted tool because of its exploitation simplicity and powerfulness.

A known technique regarding how to make CSS expression work in IE>=8 is to create a page with something like `"X-UA-Compatible: IE=7"` on the attacker's site and embed the target page as an iframe. This method works fine as long as the target page is rendered in quirks mode, which requires the target page not to contain a modern DOCTYPE like `"<!DOCTYPE html>"`.

A seemingly unknown and more powerful technique, we found in July 2014, is to use IE's CV (Compatibility View), which is apparently similar to, but different from X-UA-Compatible. In contrast to X-UA-Compatible, CV works even if the ifram'ed page of a different origin is rendered in standard mode.

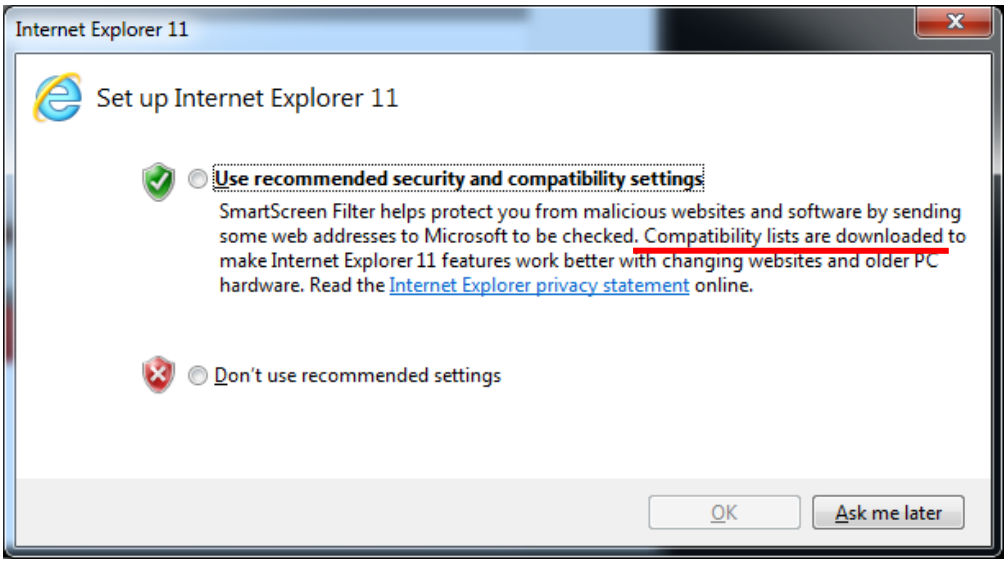
Specifically, in IE8-10, if CV is enabled in attacker's domain, CSS expression works in the ifram'ed target page even if it comes with a modern DOCTYPE. In IE11, though this technique does not make CSS expression work, this still relaxes CSS's parsing rule. For instance, CV enables multi-line string or URL literals, which is convenient for stealing data in the target page, when an attacker partially controls the page.

There are several ways for attackers to make their pages rendered in CV. The first apparent way is to trick a victim user into clicking a broken document icon in the address bar (IE8-10), or opening a menu (Compatibility View settings) in Tools menu (IE8-11). But either needs user interaction.



Another way, which does not need user interaction, is to have your domain listed in CV List (Compatibility View List). The list is an XML file hosted on a Microsoft server, and is automatically downloaded by IE if configured to do so. As its name implies, Web sites included in the list can be rendered in CV.

Probably, many of you have seen a screen below. If you chose recommended settings, CV List download feature is enabled in your environment.



The remaining question is how to have your domain included in the list. According to Microsoft [5] [6], they maintain the list based on the feedback from community. Thus there might be a chance to add your site in the list. Anyway, it can be said that domains already in the list are, in a sense, in a privileged position, while they themselves could be susceptible to attacks.

Two caveats that should be mentioned here are that you may need to reload your page to make CV take effect in its iframes, and that it works only if the target page has no explicit X-UA-Compatibility designation.

We referred to this technique utilizing CV, in a vulnerability report of XSS-related issue in IE, which was submitted to the vendor in July 2014. The vendor fixed the reported vulnerability itself by the end of 2014, but this CV issue was regarded as "By Design / Won't Fix". Thus it still works in IE as of this writing.

4. Non-stylesheet RPO attacks

In the original article authored by Heyes and other RPO-related articles, only attacks targeting CSS are seemingly mentioned.

However, as implied in the previous section, attack possibility is not necessarily limited to stylesheets. Technically, another-file-loading techniques explained in the previous section can expand the attack possibility.

As a first example, let's review a technique for Safari in 2.2.

```
■Attack URL path is:
  /member/profile_photo.php/.%2E/top.php
  → The server returns the content of "/member/top.php".

■The content returned by the server contains a script element:
  <script src="../js/jquery.js" type="text/javascript"></script>
  → Safari fetches "/member/profile_photo.php/js/jquery.js".
```

In this case, an image returned by `profile_photo.php` is executed as JavaScript.

As Matteo Carli showed in 2007 [7] (and various researchers have re-discovered later), an image file with a valid magic byte and a header structure, which is even validly renderable, can be a valid JavaScript program. This means Web sites that allow users to upload images (or other files) are susceptible to this sort of attacks.

Note that Chrome is the only browser that refuses executing an image (with image/xxx content type) as JavaScript by default. Meanwhile, recent versions of IE refuse only images with the `"X-Content-Type-Options: nosniff"` header, and Firefox, Safari and IE7 just ignore the header.

A similar XSS attack using another-file-loading technique may be possible in Ajax calls, when the URL is path-relative and the entire response is eval'ed or directly passed to innerHTML. These are sometimes seen when the supposed response is JSON data or a fragment of HTML respectively.

```
Var xhr = new XMLHttpRequest();
xhr.open("GET", "headlines.aspx", true);
```

As for innerHTML, attackers can possibly use uploaded files, JSON data and other non-HTML contents for attacks. This can work in all major browsers, and neither `"X-Content-Type-Options: nosniff"` nor `"Content-Disposition: attachment"` can prevent it.

Another target worth mentioning is a form element with a relative URL.

```
<form method="post" action="../search.php">
```

If the form has an anti-CSRF token in it, and the destination (i.e. action attribute) is controllable by an attacker, it can possibly be used for executing unwanted actions.

For instance, a user who clicked a trivial button (e.g. sign-in or search button) in a page of an attacker-supplied URL can perform unwanted actions, such as unsubscribing from the site or purchasing a product. Obviously, whether it succeeds or not depends on the request and session parameters necessary to complete these actions though.

Besides these, relative URLs can appear in various contexts. Such contexts defined in HTML and relevant specifications are listed below, just for reference.

Attribute	HTML4	HTML5	Major elements / Notes
action	X	X	form
background	X		body,table,td,th,tr
cite	X	X	blockquote,q,del,ins
classid	X		object
codebase	X		object,applet
content	X	X	meta (http-equiv="Refresh")
data	X	X	object
dynsrc			iframe,image,img,input,isindex
formaction		X	button,input
href	X	X	a,area,base,link
icon		X	menuitem
longdesc	X		img
lowsrc			iframe,image,img,input,isindex
manifest		X	html
ping			a,area (whatwg spec)
poster		X	video
profile	X		head
src	X	X	audio,embed,frame,iframe,image,img,input,isindex, script,source,track,video
usemap	X		input,img,object
value	X		param (in object elements)
xlink:href		X	(XLink spec)
url(), @import	-	-	(CSS spec)

Note that the data in the table are not comprehensive or new (it has not been updated since late 2013). Many of these may be rarely used or useless for attacks.

5. A path handling bug in CakePHP

In this section, a vulnerability of CakePHP, a Web application framework written in PHP, is outlined.

Strictly speaking, the bug outlined here is not an RPO, but it is quite similar to RPO, in terms of the attack mechanism that utilizes path manipulation to cause unexpected stylesheets loading. In the following explanation, Bookmarker application of the Cake's Quick Start is used as an example.

In Cake, developers can use a simple statement to include CSS and other files.

```
<?= $this->Html->css('base.css') ?>
```

The final HTML generated by the statement is:

```
<link rel="stylesheet" href="/bookmarker/css/base.css" />
```

As you can see, the developer's input in the statement is a relative path (file name alone), and the link in the resulting HTML is a rooted path starts with slash. This is done based on the root path determined from the request URL. A bug, which can lead to a wrong root path determination, was found in Cake\Network\Request class.

Both normal URLs and crafted one (attack URL) are shown below.

```
■Normal URLs
http://host/bookmarker/users/add
http://host/bookmarker/webroot/index.php/users/add

■Attack URL
http://host/bookmarker/webroot///index.php/users/add/{}*{a:expression(alert(document.d
omain))}/1
```

When the attack URL is given, the application fails to determine the root path, and it serves an HTML containing a link element points to the HTML itself. The mechanism is similar to that of RPO.

```
<link rel="stylesheet" href="/bookmarker/webroot//index.php/users/add/%7B%7D%2A%7Ba%3A
expression%28alert%28document.domain%29%29%7D/css/base.cs" />
```

Additionally, the stylesheet loaded by the link element reflects the vector in the path, which is underlined below.


```
<form method="post" accept-charset="utf-8" action="/bookmarker/webroot/index.php/users/
add/%7B%7D%2A%7Ba%3Aexpression%28alert%28document.domain%29%29%7D/css/users/add/{}*{a:
expression(alert(document.domain))}/css/base.css">
```

The JavaScript code inside expression() is executed on certain clients.

In response to our report, the vendor released a fix (v3.0.7 and v2.7.0-RC) that addresses the bug by adding a path canonicalization step that removes consecutive slashes [8].

A bit annoying aspect of the bug is that it is not a trivial task to automatically determine the root path correctly from the request URL. This is particularly noticeable if URL rewrite is in use, which is actually the case with Cake.

6. Conclusion

Several attack techniques that can expand exploitability of RPO are explained. The original RPO and the techniques explained in this paper are not necessarily universally applicable to any of the clients and servers, but these still work in certain environments. Therefore the risk should not be underestimated, and application developers including framework developers (and pentesters) need to keep it somewhere in mind.

As stated in other resources, the root cause of RPO is in use of path-relative URLs, which do not start with a scheme or slash. Therefore it is recommended that developers avoid using them in dynamic Web applications.

Contexts in which path-relative URLs are harmful are dependent on each application. However, in general, simple static links or images are less harmful, while those specially mentioned in this paper are more harmful.

Besides, good practices below can mitigate the risk in certain environments.

```
Add headers:
  X-Content-Type-Options: nosniff
  X-Frame-Options: DENY
  X-UA-Compatible: IE=edge

Set DOCTYPE:
  <!doctype html>
```

Not that they should normally be regarded as a second line of defense, because they are not only unsupported by some browsers, but also ineffective in some attacks.

The techniques described in the second section leverage the differences in path interpretation of servers and browsers. Concerning that, there are some specifications that standardize the URL interpretation, such as whatwg's URL specification [10], which aims to obsolete the older one, RFC3986 [9]. Though these are not directly intended for security, the more standard compliant implementations can reduce the attack window of this type.

Anyhow, the original form of RPO (self-referencing type), which uses trailing PATH_INFO, cannot be addressed by such standards. Thus some effort on the Web site developers' part is needed in any case.

7. References

[1] http://www.thespanner.co.uk/2014/03/21/rpo/

[2] http://blog.portswigger.net/2015/02/prssi.html

[3] https://soroush.secproject.com/blog/2015/02/non-root-relative-path-overwrite-rpo-in-iis-and-net-applications/

[4] http://httpd.apache.org/docs/2.2/en/mod/core.html#allowencodedslashes

[5] https://msdn.microsoft.com/en-us/library/gg699485(v=vs.85).aspx

[6] https://msdn.microsoft.com/en-us/library/gg622935(v=vs.85).aspx

[7] https://www.flickr.com/photos/matteocarli/589108973/in/photostream/

[8] https://github.com/cakephp/cakephp/pull/6747

[9] https://url.spec.whatwg.org/

[10] https://tools.ietf.org/html/rfc3986

8. Test environments

The following server / client software was used in the research.

Name	Version	OS
Apache	2.2.15	CentOS 6.3
IIS/ASP.NET	7.5 / 4.0.30319	Windows7
Tomcat	7.0.47	Windows7
WebLogic	12.1.1	Windows7
mod_wl	12.1.3 (Oracle WebLogic Server Proxy Plugins)	CentOS 6.3
IE	9.0.8112.16421 / 9.0.39 (KB3058515) 10.0.9200.17377 / 10.0.28 (KB3058515) 11.0.9600.17843 / 11.0.20 (KB3058515)	Windows7
Chrome	43.0.2357.130 m	Windows7
Firefox	38.0.5	Windows7
Safari	8.0.6 (10600.6.3)	OS X 10.10.3

9. About us

About MBSD

MBSD (Mitsui Bussan Secure Directions, Inc.) is the Japanese leading security company in managed security services, vulnerability assessment and testing, GRC (Governance, Risk, Compliance) consulting, incident response and handling, digital forensics, and secure programming training services. The MBSD services are provided by its personnel including the leading security experts in the field of secure programming, application security, penetration testing and threat analysis who have in-depth knowledge and understanding of attackers' methodologies. MBSD is working for the Internet infrastructure companies, cyber commerce and media giants, financial institutes, global enterprise, and government agencies in Japan to support their strategies against rapidly increasing threats from cyber space.

About the author

Takeshi Terada, MBSD Professional Service Dept., Senior Security Specialist, CISSP

TT-1 Building 6F, 1-14-8, Nihonbashi Ningyo-Cho, Chuo-ku, Tokyo, 103-0013, Japan
+81-3-5649-1961 | http://www.mbsd.jp/

