RPO

Friday, 21 March 2014

Relative VS Absolute

RPO (Relative Path Overwrite) is a technique to take advantage of relative URLs by overwriting their target file. To understand the technique we must first look into the differences between relative and absolute URLs. An absolute URL is basically the full URL for a destination address including the protocol and domain name whereas a relative URL doesn't specify a domain or protocol and uses the existing destination to determine the protocol and domain.

Absolute URL
https://hackvertor.co.uk/public

Relative URL
public/somedirectory

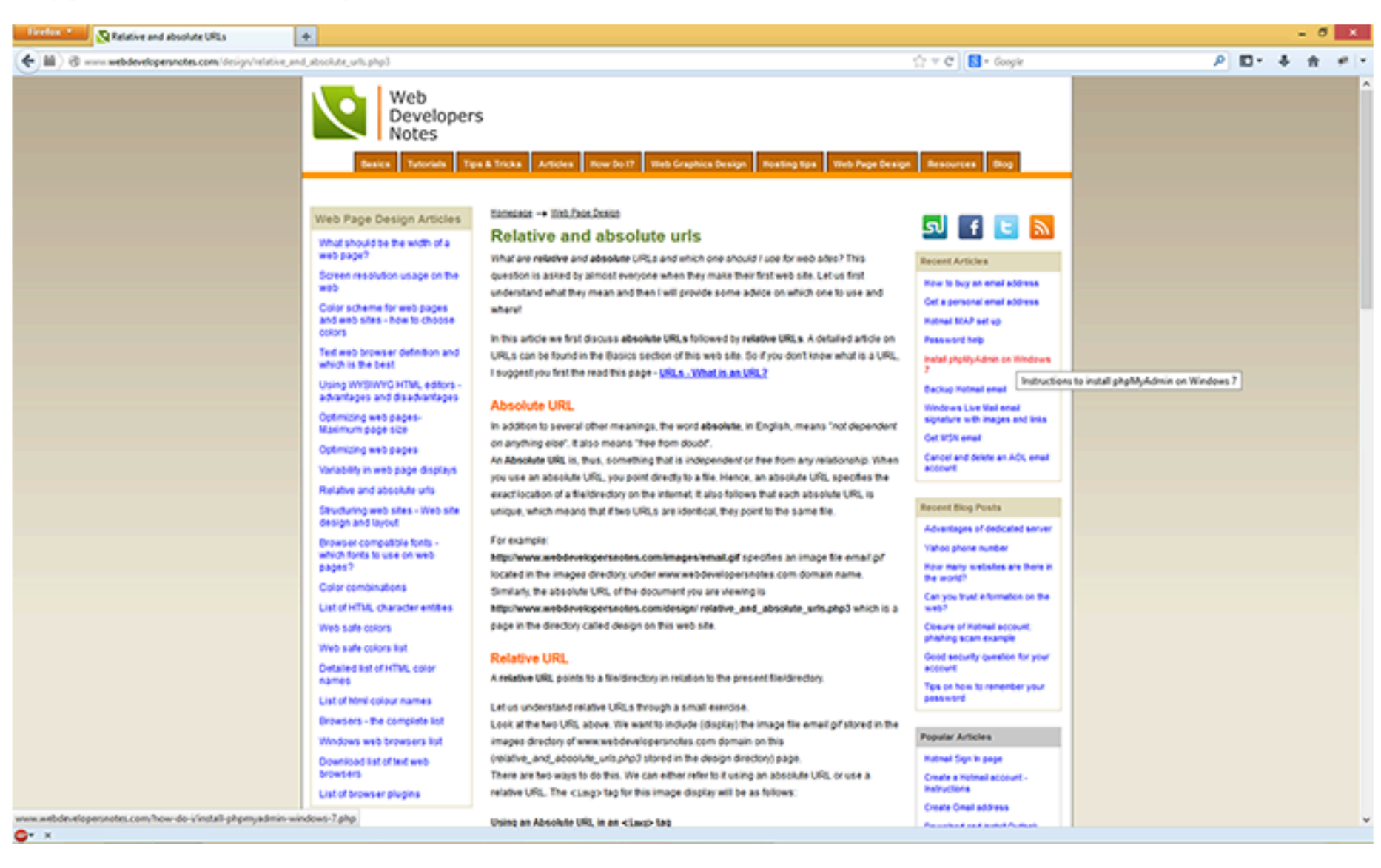
The relative URL shown will look for public and automatically include the domain before it based on the current domain name. There are two important variations of a relative URL, the first is we can use the current path and look for a directory within it such as "xyz" or use common directory traversal techniques such as "../xyz". To see how these work within markup let's take a look at a common relative URL used within a stylesheet.

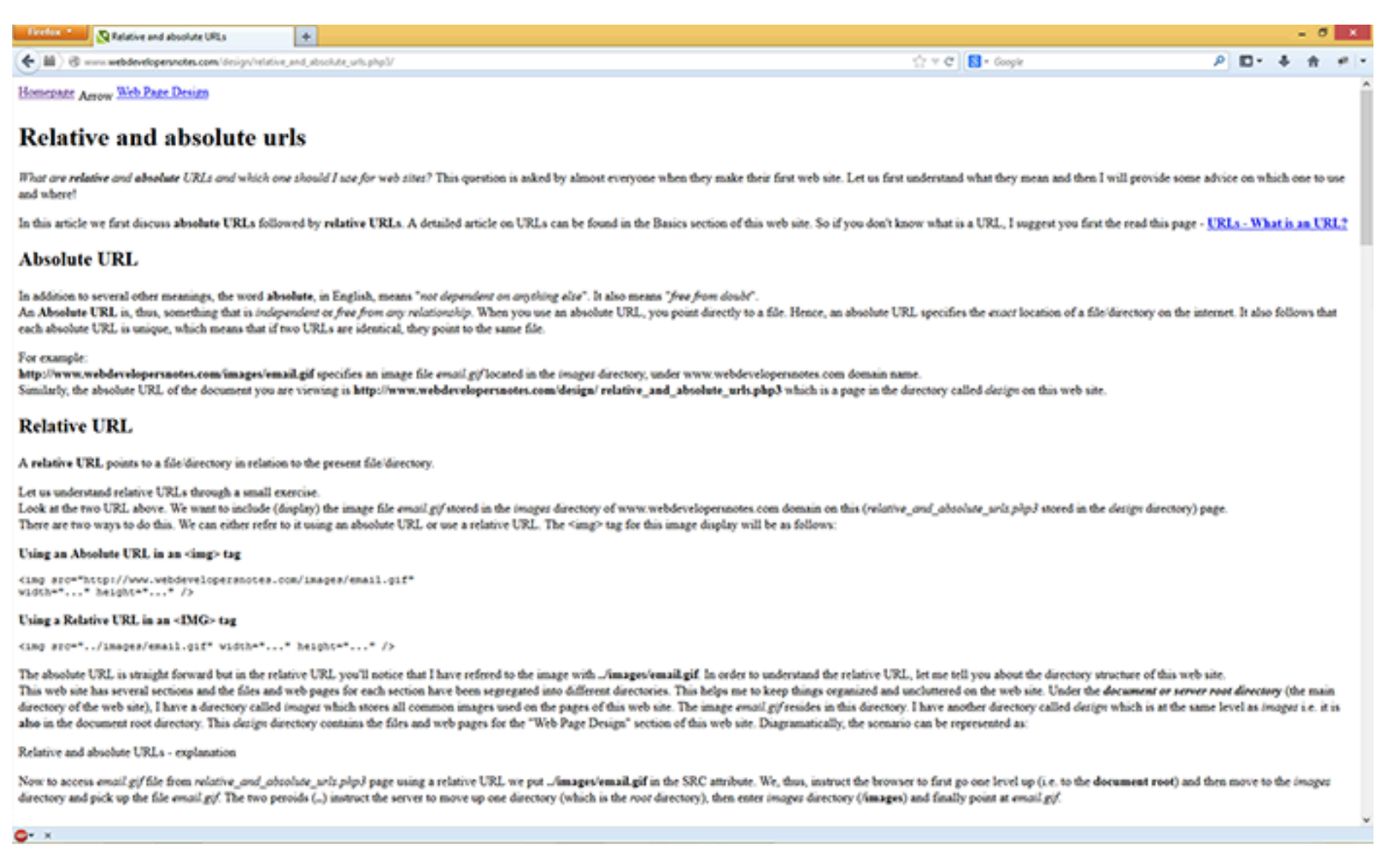
```
<html>
<head>
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
</body>
</html>
```

The link element above references "style.css" using a relative URL, depending where in the sites directory structure you are it will load the style sheet based on that. For example if you were in a directory called "xyz" then the style sheet would be loaded from "xyz/style.css".
The interesting aspect of this is how the browser knows what a correct path is since it doesn't have access to the server's file system. The answer is it doesn't. There is no way to determine a valid directory structure from outside the file system you can only make educated guesses and use http status codes to determine their existence.

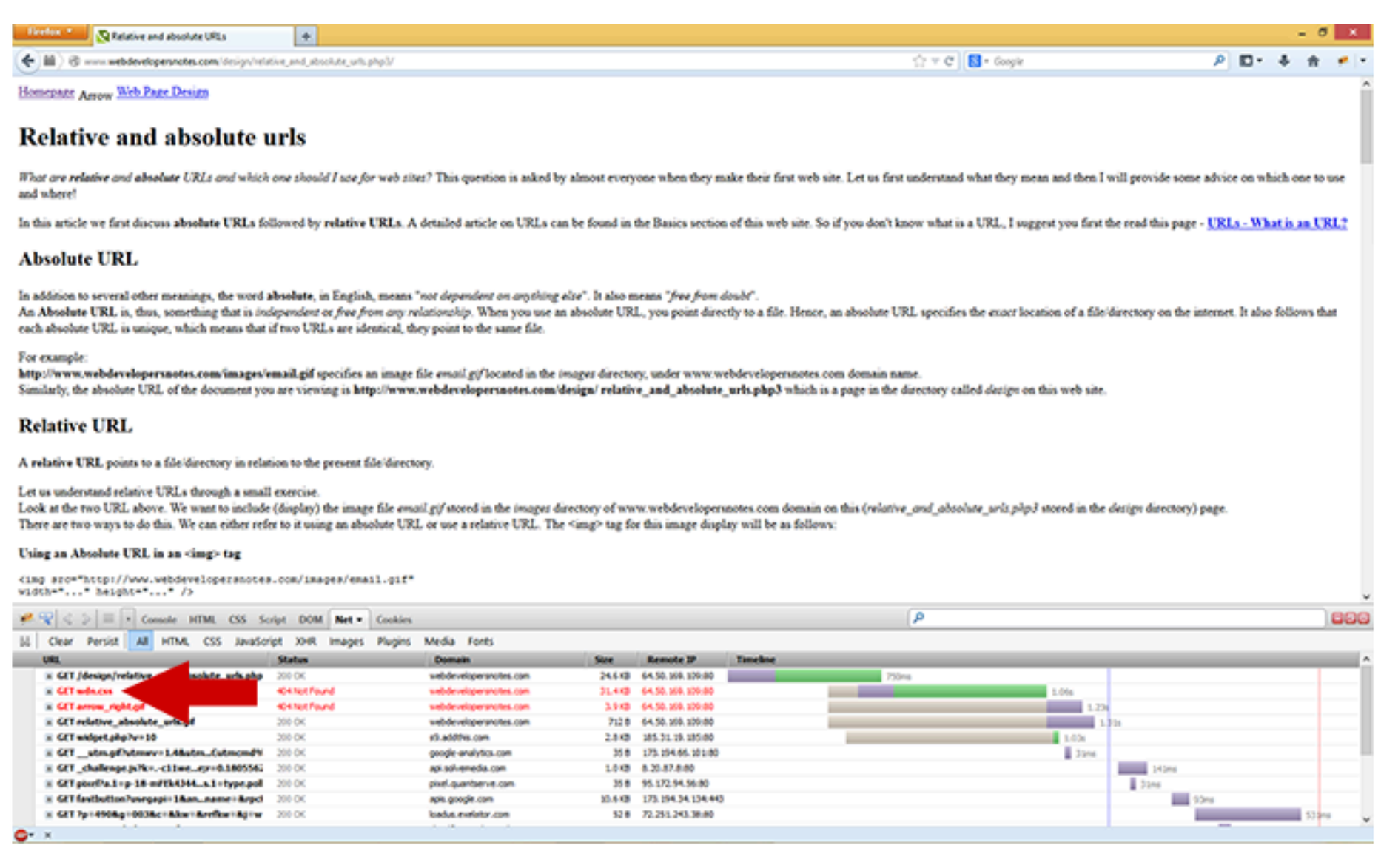
The missing styles

I noticed something interesting with relative styles, manipulating the path of the site could result in styles failing to load. It occurred to me this was a flaw in some way but the pieces of the jigsaw didn't make sense yet. How could it be exploited?





The two screenshots above show a site without manipulating with URL the styles load as expected however in the second screenshot the same site is loaded with an added forward slash and the relative style sheet does not load. Simply adding a forward slash at the end of the URL breaks the styles of the relative style. Looking in Firebug we can see the style wdn.css returns a 404 when we add the forward slash.



The screenshot shows the style sheet returning a 404 for a style that previously loaded fine without manipulating the path. If the style returns 404 maybe we can manipulate the relative URL further by changing the path. This is in essence what RPO is about, we try to change the relative URL to something we control although this post is about XSS it's worth noting that manipulating relative URLs can be done for any such URL and isn't restricted to XSS.

Quick CSS lesson

Since we are looking at manipulating a style sheet to something we control we must first understand CSS parsing in order to take advantage of it. There is an interesting piece of the CSS 2 specification that we are very interested in.
"In some cases, user agents must ignore part of an illegal style sheet. This specification defines ignore to mean that the user agent parses the illegal part (in order to find its beginning and end), but otherwise acts as if it had not been there." CSS 2 specification.
Another piece of the jigsaw is added, CSS2 ignores illegal syntax which we can use by supplying a file that contains mixed content of CSS and something else. If we can fool the CSS parsing into ignoring the illegal syntax before our intended code we can get the CSS parser to load our code. CSS selectors offer the best way to do this since an invalid selector can be ignored and all the previous illegal syntax.

Invalid code

```
}*{color:#ccc;}
```

There are two tricks to ignore illegal code both involve selectors, depending on the CSS parser a single } will work or {}. We shall look at IE compat since the parser is quite loose and supports CSS expressions. A CSS expression looks like the following:

```
*{
xss:expression(alert(1));
}
```

The first part is a global selector "*" and the { opens the selector a custom property xss is used and then the expression contains JavaScript that executes alert(1).

Self-referencing

If we can make the style sheet self-reference for the page it's on then we can use the CSS parsing to ignore the HTML and execute our custom CSS in IE compat. When a site includes a style sheet like the following:

```
<link href="styles.css" rel="stylesheet" type="text/css" />
```

We simply need to include a forward slash at the end of the URL and the style sheet will end up (if rewriting is available) loading the original page via what the browser thinks is a directory but is in fact the current page. E.g somepage.php/. Now that our style sheet is loading the web page we need to supply it with some CSS to execute, we can do this by mixing persistent data such as a first name or address think of this as both a reflective attack and a persistent attack but the persistent data contains CSS code.
To understand this it's better to show the actual structure of the page and you to see the vector itself. Imagine we have a web page with some data we control such as "first name" the web page would look like the following.

```
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
Hello {}*{xss:expression(open(alert(1)))}
</body>
</html>
```

PoC (IE ONLY): RPO example

The Meta element forces IE's document mode into IE7 compat which is required to execute expressions. Our persistent text {}*{xss:expression(open(alert(1)))is included on the page and in a realistic scenario it would be a profile page or maybe a shared status update which is viewable by other users. We use "open" to prevent client side DoS with repeated executions of alert. A simple request of "rpo.php/" makes the relative style load the page itself as a style sheet. The actual request is "/labs/xss_horror_show/chapter7/rpo.php/styles.css" the browser thinks there's another directory but the actual request is being sent to the document and that in essence is how an RPO attack works.

Further RPO attacks

You might wonder if the RPO attack is restricted to just relative URLs like "styles.css" the answer is no, it's possible to attack URLs such as "../../styles.css" but in this case we need to provide levels of fake directories until the styles are loaded from the current document. "../" means look above the current directory; we need three levels of fake directories.

```
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<link href="../../styles.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
Hello {}*{xss:expression(open(alert(1)))}
</body>
</html>
```

PoC: RPO example 2

This time because the relative URL is looking for a directory twice above the current directory we make a request of "/labs/xss_horror_show/chapter7/rpo2.php/styles.css" this means that you could also target a file in a different directory but in this case we pointed it back to the original html file. Note we could have done just rpo2.php/// but I provided the text of fake directory for clarity.
There are other variants such as using the @import command which is useful if length or characters are limited. Using the "}" to ignore the HTML again followed by an @import statement works perfectly fine on IE even though technically it's invalid syntax to use an import statement in this way.
RPO isn't restricted to IE, we can use the technique on other browsers but JavaScript isn't supported in CSS on Chrome, Firefox, Opera or Safari. Another restriction is that a doctype cannot be included on the target document since this causes the CSS's parsers to stop parsing the HTML file on non-IE browsers.

```
<html>
<head>
<link href="../../styles.css" rel="stylesheet" type="text/css"
/>
</head>
<body>
Hello {}*{color:#ccc;}
</body>
</html>
```

PoC: RPO example 3

The document above changes the colour of the text to grey and works on every browser. It works in the same way as the previous PoC but this time uses pure CSS and no expressions. If a doctype was included in the document it would fail on every browser except if IE was in compat mode.
RPO attacks work on any type of document, it's possible to change the target of image files for example but because the image files look for specific strings at the start of the file and the end result is only an image it makes RPO attacks less useful in these circumstances.

Reflected RPO

If the URL is outputted on the page we can send the XSS vector via the path. The following PHP example shows the URL being outputted on the page.

```
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
Hello <?php echo $_SERVER['PHP_SELF']?>
</body>
</html>
```

There is a relative URL here and "echo $_SERVER['PHP_SELF']" outputs the current URL of the page requested. We can exploit this by providing some CSS as part of the path since the relative URL will be loaded with our injection and then the CSS will be loaded from the HTML. On some configurations PHP_SELF will truncate the path information, in this instance PATH_INFO can be used.
PoC: RPO example 4

Summary

I consider relative URLs harmful since you cannot rely on the browser to correctly determine the correct directory and when used with so called "pretty URLs". Pretty much everyone who has used relative URLs will be open to this type of attack if the path information is outputted or there is some persistent data that an attacker can manipulate. I recommend absolute URLs should be used throughout a site or relative URLs that begin with a forward slash since this is the only type of relative URL that isn't vulnerable to a RPO attack because it starts at the document root.

Update..

It's worth noting that a relative root url isn't vulnerable to this sort of attack since the directory is take from the highest point in the structure and I think can't be influenced the way a normal relative url can.

The entry 'RPO' was posted on March 21st, 2014 at 9:09 pm and last modified on September 18th, 2014 at 8:46 pm, and is filed under Uncategorized. You can follow any responses to this entry through the RSS 2.0 feed. Both comments and pings are currently closed.

Comments are closed :(too much spam. If you want to contact me about any article please email or tweet me.

« Sandboxed jQuery | Bypassing the XSS filter using function reassignment »

Search...

Inspiration

Alex Inführ
Arshan
Ascetik
beford
Billy Rios
Chris Weber
David Ross
Eric Lawrence
hackademix
Hackvertor
Halvar Flake
Jesse Ruderman
Joe Walker
John Resig
Kuza55
maliciousmarkup
Manuel Caballero
Matt Presson
Miscoded
nihilogic
PHPIDS
pro.grammatic
Reiners
rgaucher
rvdh
Sirdarckcat
sla.ckers
Soroush Dalili
Stefan Esser
Stefano Di Paola
Thornmaker
tssci-security
ush.it
Web Reflection
xorl
Yosuke HASEGAWA

Recent Comments

Gareth Heyes on Sandboxing and parsing jQuery in 100ms
Ariya on Sandboxing and parsing jQuery in 100ms
Gareth Heyes on Sandboxing and parsing jQuery in 100ms
Rick on Sandboxing and parsing jQuery in 100ms
Gareth Heyes on MentalJS Sandbox/Parser

March 2014

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

« Feb   Apr »