Persistent Cookies and DNS Rebinding Redux

Persistent Cookies and DNS Rebinding Redux - Author not stated, ha.ckers.org.

- Published: date not stated
- Original: <http://ha.ckers.org/blog/20090120/persistent-cookies-and-dns-rebinding-redux/>
- Preserved from: http://ha.ckers.org/blog/20090120/persistent-cookies-and-dns-rebinding-redux/ (manual-import) on 2026-09-10
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

In an attempt to clarify my post on the dangers associated with persistent cookies and DNS rebinding, I'd like to give a simple scenario and then describe solutions. Let's say there is an intranet website called intranet.exploitable.com that resolves to 10.10.10.10, and there is an attacker website called www.attacker.com that resolves to 222.222.222.222. Now let's say intranet.exploitable.com typically sets a cookie that has also has a known XSS vulnerability in it (could be known because the attacker knows what sort of open source software is used internally, or they were once a contractor or whatever...). Now let's also assume that the website is not SSL, as most aren't and it would mess up the attack with a mis-match SSL error.

Okay, so the victim user visits www.attacker.com who sets the same cookie as something like this:

```
Set-Cookie: last-visited=<script>alert("XSS")</script>; path=/
```

Then the user shuts down their browser, or the attacker forces a browser shutdown through any one of the dozens of browser DoS scripts out there. Eventually the user goes back to www.attacker.com, but this time, the site changes it's DNS to point to 10.10.10.10. Because the browser was shut down, the DNS for www.attacker.com is now allowed to be rebound to the new IP address, which happens to be the IP address of intranet.exploitable.com. The user now visits that site with the XSS exploit in their cookies, with the incorrect host header:

```
Host: www.attacker.com
```

However, because most sites don't care about host headers, the request is still parsed by intranet.exploitable.com's website. The XSS is now running there. While this wouldn't allow the attacker to log into their account, it would allow them to "see" what is running on the victim's

intranet website, by using an XSS shell. Although this attack may take a while, it's not that difficult, compared to a lot of other rebinding attacks.

Now in terms of mitigation, there's a whole host of things you can do if you happen to run intranet.exploitable.com. Firstly, using SSL would stop this attack because of the SSL to hostname mis-match. Secondly, not allowing any unknown host header to be sent would stop the incorrect host header from being processed. Using client side protections like LocalRodeo would stop the intranet from being contacted as well. Lastly, making sure that *all* cookies are removed upon each shut down of the browser would stop the attacker from being able to re-use their cookies after having forced the victim's browser to shut down. I hope all that was a lot more clear.

Technical comments

Comments below are quoted from the archived source and retain the commenters' claims and qualifications.

Comment 1

Nick Says:

January 20th, 2009 at 12:50 pm

You can also make sure that your web application isn't vulnerable to XSS attacks sent via cookies.

Comment 2

Adam Barth Says:

January 20th, 2009 at 1:17 pm

It might be more fun to think about a persistent XSS vulnerability in the intranet server. Now the attacker's script is stored in the site's database and can later be triggered by navigating the user's browser to http://intranet.exploitable.com/. In this variation, the attacker gets to run his XSS shell with the user's privileges on the vulnerable site.

Comment 3

RSnake Says:

January 21st, 2009 at 3:30 am

@Nick - Sure, I was saying the other fixes work in the case where not every use case for every cookie on every page is known. There are some sites that have more cookies than even the browser can handle - it would be nearly impossible for them to test all of them on every page. Sometimes the simplest solutions are easily the hardest to implement effectively. ;)

@Adam - you're the second person to say this, but XSS doesn't have to be persistent to run a shell (I got another similar response in email). The victim just has to stay on whatever page you landed them on for long enough to do the damage necessary (sometimes only fractions of a second - sometimes hours).

Comment 4

Adam Barth Says:

January 21st, 2009 at 11:37 am

@RSnake - The difference is that, with reflection, your XSS shell is running as www.attacker.com, but with a persistence, your XSS shell is running as intranet.exploitable.com. Running under the victim's host name is much more powerful than running under the attacker's host name because now you have access to the intranet.exploitable.com cookies (which likely include an auth cookie) and the intranet.exploitable.com password database. In addition, you can create a perfect phishing page because the location bar contains the trusted URL.

Comment 5

RSnake Says:

January 21st, 2009 at 12:31 pm

I understand what the value difference is, but I think if that were the case, it would be far more likely that you'd just use CSRF to submit the persistent XSS - no DNS rebinding necessary. DNS Rebinding gets around the fact that you may not have any other exploits there other than the single reflected XSS in the cookie.
