Interesting JForum vulnerabilties and the ESAPI WAF

转载

cnbird2008 于 2009-11-20 11:26:00 发布 986 收藏
文章标签： application freemarker security token templates email

The application I beat up for the ESAPI WAF preso at OWASP AppSec DC was JForum. It's awesome, free, open source forum software that is quite popular (CBS, EA and the Ukrainian government seem to like it). That aside, it's got serious security problems. I disclosed these problems to them, um, around a month ago or so, and some of these vulns are interesting for one reason or another, so I thought it'd be good to highlight a few here.

Vuln #1: Hijack accounts with "Forgot Password" token prediction

You don't have to be Nate Lawson to discover this obvious security flaw, which may from a blind test may appear secure.

The "forgot password" feature suffers from a critical design error. The application allows users to automatically reset their password through a "lost password" form. That form only requires a user to enter an email address or username. When the form is submitted, the application sends an email containing a token to the associated user's email address. When the user clicks on the link with the token in it from the email, the server will then allow the user to reset their password through a form. In this design, the token is acting as a temporary password for the user. Therefore, if an attacker can predict the value of the token the application will generate, they will be able to reset account passwords for other users.

Unfortunately, this scenario is possible. The following code is from UserAction.java, starting at line 671:

```
public User prepareLostPassword(String username, String email)
...
String hash = MD5.crypt(user.getEmail() + System.currentTimeMillis());
...
um.writeLostPasswordHash(user.getEmail(), hash);
user.setActivationKey(hash);
```

As can be seen in the code snippet, the secret token is an unsalted hash of the user's email address and the number of milliseconds since the epoch. Neither of these pieces of information qualify as secrets in a strong cryptosystem. All that is needed to reset a user's password to a password of the attacker's choosing is the email address of the victim and the ability to generate a few thousand requests.

An exploit was made to demonstrate this vulnerability. It's currently tuned to attack a local development environment, but it can be used to attack any site by changing a few variables (and adding a time difference offset). Since the application leaks the server's time in several places, it's possible to increase the efficiency of the exploit so that remote systems only require a few hundred packets. The exploit is available here.

Vuln #2: The first and last interesting XSS flaw in the world

JForum has a reflected XSS flaw whose exploitation is uniquely non-trivial. To start: in my development environment, this URL causes an alert box to pop up, containing my session cookies:

http://localhost:8080/JForum/nonexistent.page?
module=js&action=list&js=../admin/admin_welcome.htm%00<script>alert(1)</script>

This is an interesting story, and the reason this fires is complex, as far as XSS goes. First, that type of URL won't normally be found in JForum. The typical URL structure is RESTful. For example, the URL to list the recent posts in a category would look something like this:

http://localhost:8080/JForum/recentTopics/list.page

This is parsed by JForum into a "module" and an "action". The "recentTopics" substring represents the "module" of this URL, and "list", the "action." However, there is an alternative and possibly legacy representation for URLs that is honored by JForum, where URLs look like this:

http://localhost:8080/JForum/servlet.page?module=recentTopics&action=list

The "servlet" substring in this URL is arbitrary, since JForum's main engine catches all requests aimed at "*.page" and handles them identically based on parameters. This will end up making an attack signature difficult, as will be shown later.

Normally, no user input in JForum is rendered without encoding. The text being output in this example is being done so by Freemarker, the templating system used by JForum. Because the template to execute is supplied by the user (the 'js' parameter), the user can choose any file on the target filesystem. If the file is not a pre-approved template, the application will error without doing anything necessarily beneficial to the attacker. The only alternative, then, is to provide a template that the application won't have correct contextual data for. This will happen, for instance, when a normal user attempts to view an admin template. When the expected data isn't found this will inevitably cause a problem, after which Freemarker will print an error message that contains the filename being executed, along with other data which is not controlled by the user.

So, the only way for this to be useful is if the filename contained an attack. How could the filename be both an attack and a valid template location? This is about the time a normal developer would cry theoretical and reject the vulnerability. Unfortunately for them, by supplying the null byte followed by a traditional XSS payload after the template location, we can make both JForum/Freemarker and the attacker happy. When JForum/Freemarker look up the filename, the file system will acknowledge that the file exists and is a pre-approved template. However, when the filename is printed out by Freemarker after the error occurs, it will echo the entire filename parameter, not just the part of the filename understood by the lower level APIs. Because the attack is part of the extended filename, it gets echoed to the browser, and the JavaScript fires.

This vulnerability could not exist without 3 combined failures of the OWASP Top 10: flawed error handling (printing detailed error information), direct object references (specifying arbitrary templates in the URL) and XSS. This is a credit to the code of Rafael Steil, the maintainer of JForum, who otherwise makes a (relatively) security conscious product.

Note: One might guess that the ability to specify arbitrary template constitutes an elevation of privileges. The templates are read-only and are publicly available, since the forum software is OSS. For it to be useful, the application would first have to queue the relevant admin information into memory before processing the template. This would require a separate vulnerability.

For giggles, though, there are other XSS flaws that are much more vanilla:

http://localhost:8080/JForum/jforum.page?
module=user&action=recoverPassword&hash=foo%22%3E%3Cscript%3Ealert%28document.cookie%29%3C/

Now, how do we fix these in the ESAPI WAF?

These were not the vulnerabilities I used in my demo because they were not very clean to fix with a WAF (of any kind, not just mine). Unfortunately, they were two of the most serious vulnerabilities.

The account hijacking vulnerability is not possible to stop with a WAF without IP-based throttling, which is butthurt. This is the case because the attack is possible unauthenticated, so there's no way to differentiate between legitimate "forgot password" resets and attacks. Come to think of it, the WAF could also just prevent all access to that feature until you get the code fix in. Does that count?

It's possible but annoyingly inefficient to signature the complicated XSS with a simple virtual patch rule. First you'd have to signature a substring of the URI to detect that particular module being executed and then also signature the "js" parameter to detect non-alphanumerics. Given the fact that JForum can have multiple URLs for executing the same functionality, this protection doesn't give a whole lot of assurance.

But wait, there's less (security)

There are a ton more problems, like CSRF, unchecked redirects, no frame-breaking code and more. For the full writeup that I sent to the developers, click here.