Abusing Locality in Shared Web Hosting

Nick Nikiforakis

About me

- PhD student at KUL
- Applied security research
  - Low-level countermeasures for unsafe languages
  - Web application security
- Published in academic/industry and hacking conferences
- http://www.securitee.org

DistriNet RESEARCH GROUP

KATHOLIEKE UNIVERSITEIT LEUVEN

In one sentence…

- Two novel server-side session attacks against Web applications hosted in a shared-hosting environment, which target a Web application's logic instead of authenticated users
  - Bypass authentication mechanisms
  - Elevate privileges
  - Conduct, previously impossible, attacks

Roadmap

- Shared Hosting
- Session Identifiers
- Session Attacks
  - Standard (client-side)
  - Session Snooping, Session Poisoning (server-side)
- Who is affected
- Existing Protection mechanisms
- Conclusion

Shared Hosting

- 124,953,126 active domains[1]
  - 121,121 registered today
- Hosting companies
  - Shared Hosting
  - Virtual Dedicated Hosting
  - Dedicated Hosting

[1] http://www.domaintools.com/internet-statistics/

Shared Hosting Prices

- Shared Hosting
  - Starting at 3.64 Euro/month
- Virtual Dedicated Hosting
  - Starting at 21.89 Euro/month
- Dedicated Hosting
  - Starting at 45.97 Euro/month

$$\begin{tikzcd}
\text{Shared Hosting} \arrow[d, bend left=60, "6X"] \\
\text{Virtual Dedicated Hosting}
\end{tikzcd}$$

Shared Hosting

- Many users share one server
- Typically:
  - 1 Virtual Host Setting/User
  - User is confined to a small number of directories
  - All web applications run with the privileges of the Web Server

Downsides of Shared Hosting

- More Limits
- Less Control
- Less Performance
- <u>LESS SECURITY!</u>

Sessions

HTTP & HTTPS

- The two workhorse protocols are by design stateless
  - No native-tracking mechanism provided
  - Inability to enforce access control
- Mechanisms
  - HTTP Authentication
  - Client-side SSL certificates
  - Session identifiers

Session Identifiers

- Generate pseudo-random identifier (token) and bind that with a specific user
- Give this token to the user
- Every time that the user visits the page, make the distinction based on that token

- Indispensable feature of the modern WWW
  - All Web-programming languages support it

Session Cookie

- Ways to communicate the session identifier to the user:
  - As a cookie
    - PHPSESSID=qwertyuiop;
  - As a GET parameter
    - http://www.mysite.com/index.php?ID=qwertyuiop

Well-known session attacks

- Session Hijacking
  - Through XSS
    - XSSed contains more than 300,000 records
  - Sniffed Traffic
    - Open WiFi, TOR Exit nodes
    - Most recent-tool, FireSheep
- Session Fixation
  - Get a valid session
  - Let the user populate it
  - Then use it again

Sessions and the Server

Behind the scenes

- session_start(), creates a file that will contain all the values that the programmer will set in the $_SESSION[] array

- The filename consists of a standard prefix and the session_id itself
  - – Set-Cookie: PHPSESSID= qwertyuiop
  - – Filename: sess_qwertyuiop
  - – Stored in the default session store
    - /tmp, /var/lib/php5,…

What does the session file look like

- $_SESSION['loggedin'] = 1;
- $_SESSION['user'] = "admin";
- $_SESSION['num'] = 4.5;

- loggedin|i:1;
- user|s:5:"admin"
- num|d:4.5

Behind the scenes

User With Session

GET /index.php

Cookie:

PHPSESSID=12345678

….

session_start() $\longrightarrow$

Open file: $Session_store/$Prefix_12345678

Populate $_SESSION[] array with values from this file

Facts…

- By default, all PHP scripts share a common session store
- The session file accessed by PHP is based on the session id provided by the user
- A Web application can’t distinguish between sessions that it created and sessions that other applications created

Results…

An attacker with a single malicious PHP script can:

1. force a co-located web application to use sessions that it didn’t create
2. Open session files that he didn’t create and make arbitrary changes

Results…

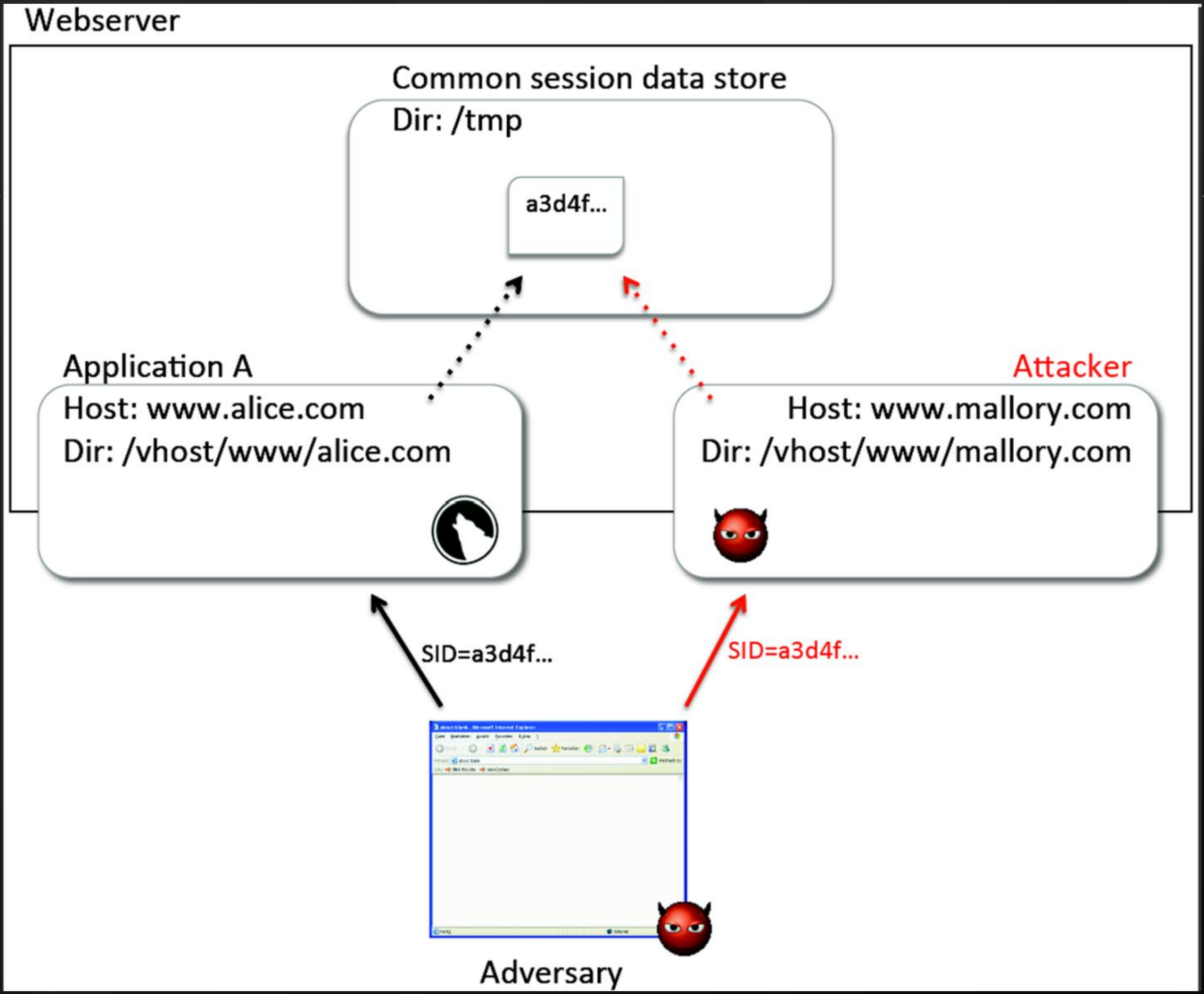
An attacker with a single malicious PHP script can:

1. force a co-located web application to use sessions that it didn't create

2. Open session files that he didn't create and make arbitrary changes

Session Poisoning

Session Snooping



```
if (isset(
$_SESSION['isadmin'])
){
   //Administrative panel
[...]
}
```

```
$_SESSION['isadmin'] = True;
```

Session Poisoning…

1. An attacker creates a new session
2. Populates this session with common variable names
   - $_SESSION['loggedin'] = 1
   - $_SESSION['isadmin'] = 1
   - $_SESSION['user'] = "admin"
   - $_SESSION['userid'] = 0
   - …

Session Poisoning…

3. Forces the session cookie to all of the websites/web applications located on the same server
4. If an application uses the same naming of variables then the attacker can circumvent the logic of the application
   – E.g, if (isset($_SESSION['isadmin']))

Session Snooping

1. The attacker visits a co-located website, creates an account and does an “exhaustive” browsing of the website
2. He prints out his session identifier
3. He instructs his own scripts to load the session file with the session identifier of the website in question
   i. Legitimate operation of session_id()

Session snooping…

4. He looks at the values that the website has set in the session identifier
5. He edits/adds values which will enable him to elevate his rights
   – $_SESSION[‘userid’] = 45;

Session snooping…

4. He looks at the values that the website has set in the session identifier
5. He edits/adds values which will enable him to change/elevate his rights
   - $_SESSION['userid'] = ~~45~~;
   - $_SESSION['userid'] = 44;

Attacker Methodology

- Mass Attacks
  - Obtain list of websites located on the same physical server as you
  - Create a session and set many common keywords
  - Browse all the different websites, always forcing the session cookie that you created

Attacker Methodology

- Specific targets
  - Place yourself on the same server as your victim
  - Browse their website extensively and then load their session in your PHP snooping script
  - Change values at will
  - Reload page

DEMO!

- Hopefully…

Attacks made possible

- Expanding the attack surface
  - Programmers trust their own input
  - SQL, XSS, Local/Remote file inclusion…

```
SELECT fname,lname,email from users where
userid = $_SESSION['userid'];
```

```
$_SESSION['userid'] = '-1 UNION ALL
SELECT…';
```

Attacks made possible

- Evading Web application firewalls
  - Session values that are used in SQL requests are never in the URL or body of the request
- Evade logging
  - Attack vector is not present in the attacker's request, thus it will never show in any kind of logging

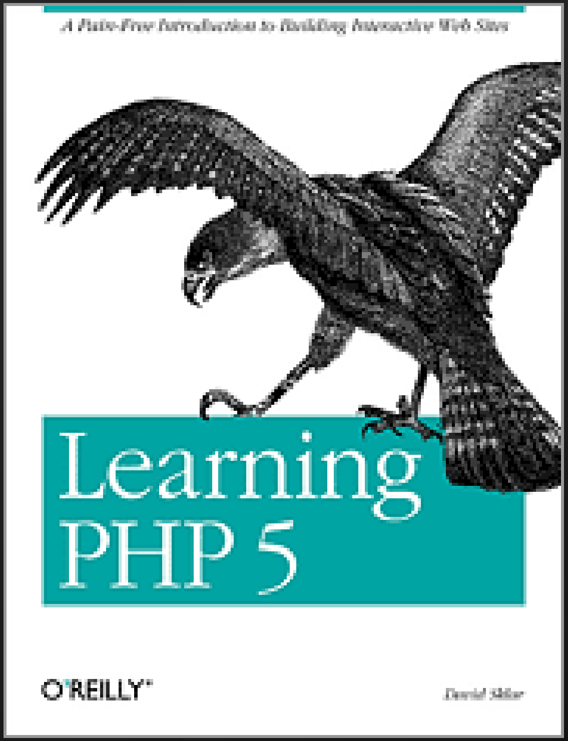
Roadmap

- Shared Hosting
- Session Identifiers
- Session Attacks
  - Standard (client-side)
  - Session Snooping, Session Poisoning (server-side)
- <u>Who is affected</u>
- Existing Protection mechanisms
- Conclusion

Who is affected?

- Everyone hosted on a shared hosting environment who is not actively protecting their sessions
  - Open source applications
    - forum-software, picture galleries, web admin panels, CMS …
  - Custom scripts

Teaching Programmers…



Chapter 8:
“Sessions work great with no additional tweaking….”

Common session stores

- How popular is the use of common session stores?
- Crawl phpinfo pages on 500 websites
- 89.71% kept the default values
  - /tmp
  - /var/lib/php4
  - C:\PHP\sessiondata

Case Study: CMS

- Content Management Systems
- Enable non-programmers to create professional, dynamic and powerful websites

CMS: Results

- 9 out 10 used sessions to maintain state
- 2 out of 9 used the default PHP session functionality…
  - Concrete5 & WolfCMS
  - 22.2% Vulnerable

- The non-vulnerable ones used the database to store their sessions

Roadmap

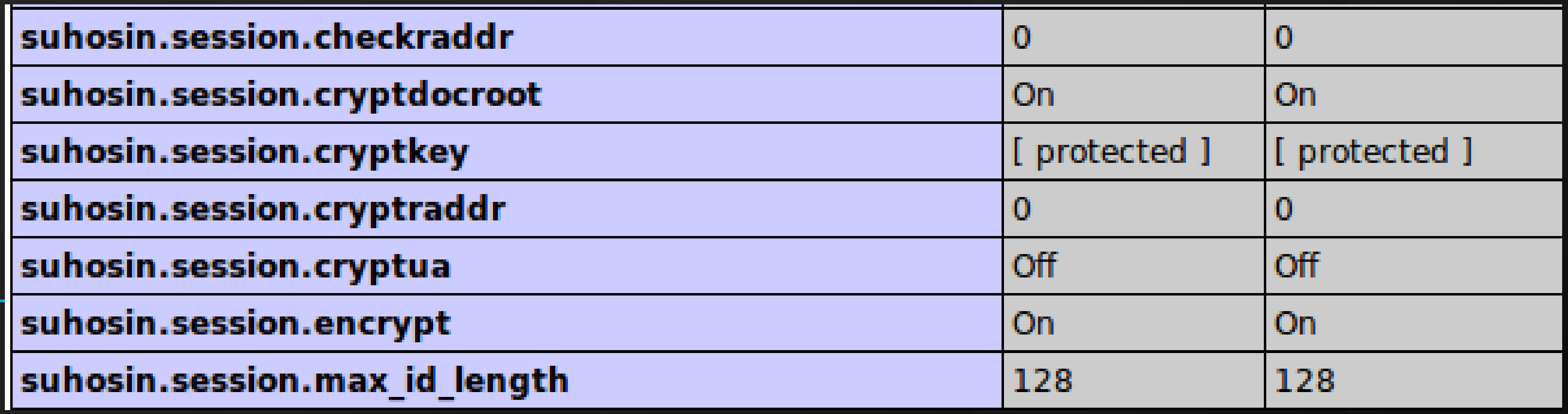
- Shared Hosting
- Session Identifiers
- Session Attacks
  - Standard (client-side)
  - Session Snooping, Session Poisoning (server-side)
- Who is affected
- Existing Protection mechanisms
- Conclusion

Suhosin

- Suhosin is an advanced protection system for PHP installations. It was designed to protect servers and users from known and unknown flaws in PHP applications and the PHP core.
  - Patch to protect core
  - Extension to protect applications

수호신

Suhosin Session Defaults



suhosin.session.checkraddr	0	0
suhosin.session.cryptdocroot	On	On
suhosin.session.cryptkey	[protected]	[protected]
suhosin.session.cryptraddr	0	0
suhosin.session.cryptua	Off	Off
suhosin.session.encrypt	On	On
suhosin.session.max_id_length	128	128

Session data can be encrypted transparently.
The encryption key used consists of this user defined string (which can be altered by a script via ini_set()) and optionally the User-Agent, the Document-Root and 0-4 Octects of the REMOTE_ADDR.

Other server solutions

- suEXEC, suPHP, fastcgi…
- One common goal
  - Run applications with specific user privileges instead of “nobody” web user
  - We can no longer open other peoples’ session files and snoop around (Session Snooping)
  - 16-35x overhead
  - But?

Can we go around these?

- If the session store is still common, yes ☺
  - Create and poison session
  - Change permissions of session file to 0777
  - Force site to use the specific session id
    - This will work because your file is available to all other users

Roadmap

- Shared Hosting
- Session Identifiers
- Session Attacks
  - Standard (client-side)
  - Session Snooping, Session Poisoning (server-side)
- Who is affected
- Existing Protection mechanisms
- <u>Conclusion</u>

Take away…

- In an shared hosting environment where:
  - Each Web application runs as a different user
    - Isolation enforced by the OS
  - A Web shell won't help you touch other Web applications
  - Abusing the common session storage to perform Session Poisoning and Session Snooping, will

Conclusion

- Session management functionality of PHP was NOT designed with shared hosting in mind…
- Two novel server-side attacks against session identifiers
  - Bypass authentication
  - Impersonate users
  - Perform, previously impossible, attacks

Thank you

- Questions?

Contact:
nick.nikiforakis [AT] cs.kuleuven.be

http://www.securitee.org
http://demo1.cz.cc
http://sessionattacker.cz.cc