Secunia
Stay Secure

PHP Object Injection in Joomla...questo sconosciuto!

Napoli, 12 Ottobre 2013

About Me

- Egidio Romano (aka EgiX)
- Web Application Security Researcher dal 2007
- Security Specialist presso Secunia dal 2013
- https://www.linkedin.com/in/romanoegidio

Agenda

- PHP Object Injection
  * La funzione "unserialize"
  * OOP in PHP: autoloading e metodi magici
- PHP Object Injection in Joomla
  * CVE-2013-1453 e CVE-2013-3242
  * Attacchi SQL Injection e Denial of Service

PHP Object Injection

- Classe di vulnerabilità scoperta da Stefan Esser nel 2009: "Shocking News in PHP Exploitation"
- Successivamente rivisitata sempre da Esser durante BlackHat USA 2010: "Utilizing Code Reuse/ROP in PHP Application Exploits"
- Ma da allora nessun'altra notizia a riguardo...

PHP Object Injection

...da Novembre 2011 ho iniziato a svolgere ricerca su questo tipo di vulnerabilità:

- CVE-2012-0694: SugarCRM <= 6.3.1
- CVE-2012-0911: Tiki Wiki CMS <= 9.3
- CVE-2012-5692: IP.Board <= 3.3.4
- CVE-2013-1465: CubeCart <= 5.2.0

La funzione "unserialize"

Il linguaggio PHP implementa un meccanismo di serializzazione che permette di convertire in formato stringa qualsiasi tipo di variabile utilizzando la funzione "serialize". A partire da questa rappresentazione sotto forma di stringa è possibile ricostruire la variabile a run-time attraverso la funzione "unserialize".

La funzione "unserialize"

```php
<?php
$serialized_data = serialize(array('Math', 'Language', 'Science'));
echo  $serialized_data;
// Unserialize the data
$var = unserialize($serialized_data);
// Show the unserialized data
var_dump($var);
?>
```

Output:

```
a:3:{i:0;s:4:"Math";i:1;s:8:"Language";i:2;s:7:"Science";}
array(3) { [0]=> string(4) "Math" [1]=> string(8) "Language" [2]=> string(7) "Science" }
```

La funzione "unserialize"

```php
<?php

class Esempio
{
   public $a = 1;
   public $b = 'test';
}

$obj = new Esempio();
echo serialize($obj);

?>
```

Output:

```
O:7:"Esempio":2:{s:1:"a";i:1;s:1:"b";s:4:"test";}
```

La funzione "unserialize"

Warning

Do not pass untrusted user input to **unserialize()**. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

http://php.net/manual/en/function.unserialize.php

OOP in PHP: autoloading

Molti sviluppatori che producono applicazioni orientate agli oggetti creano un file sorgente PHP per ogni classe definita. Uno dei più grossi inconvenienti di questo approccio è la necessità di mantenere una lunga lista di inclusioni all'inizio di ogni script (un'inclusione per ogni classe). In PHP 5, ciò non è più necessario, in quanto è possibile definire una funzione __autoload() che viene automaticamente invocata in caso si stia cercando di usare una classe/interfaccia che non sia stata ancora definita.

http://php.net/manual/it/language.oop5.autoload.php

OOP in PHP: autoloading

Questo esempio cerca di caricare le classi ***MyClass1*** e ***MyClass2*** rispettivamente dai file *MyClass1.php* e *MyClass2.php*.

```php
<?php

function __autoload($class_name)
{
    include $class_name . '.php';
}

$obj  = new MyClass1();
$obj2 = new MyClass2();

?>
```

OOP in PHP: metodi magici

I metodi magici sono sintatticamente riconoscibili attraverso il doppio underscore ("__") iniziale. Essi sono metodi "speciali" che PHP invoca implicitamente ed automaticamente in particolari circostanze, ovvero al verificarsi di un evento.

http://php.net/manual/it/language.oop5.magic.php

OOP in PHP: metodi magici

Nome del metodo	Invocato quando…
__construct	viene creata una nuova istanza della classe
__destruct	viene distrutta un'istanza
__call	viene richiamato un metodo inaccessibile (nel contesto di un oggetto)
__callStatic	viene richiamato un metodo inaccessibile (nel contesto statico)
__set	viene usata in scrittura una proprietà inaccessibile
__get	viene usata in lettura una proprietà inaccessibile
__sleep	viene richiamata la funzione "serialize" con un'istanza
__wakeup	viene richiamata la funzione "unserialize" con un'istanza
__toString	viene utilizzato un oggetto come una stringa

Un semplice esempio

```php
<?php

class Esempio
{
    public $cache_file;

    function __construct($file)
    {
        $this->cache_file = $file;
    }

    function __destruct()
    {
        if (file_exists($this->cache_file)) @unlink($this->cache_file);
    }
}

$user_data = unserialize($_GET['data']);

?>
```

```
http://[host]/vuln.php?data=O:7:"Esempio":1:{s:10:"cache_file";s:11:"/etc/passwd";}
```

PHP Object Injection in Joomla

- CVE-2013-1453: questa vulnerabilità si trova nel plugin di sistema "highlight", all'interno del file /plugins/system/highlight/highlight.php

- CVE-2013-3242: questa vulnerabilità si trova nel plugin di sistema "remember me", all'interno del file /plugins/system/remember/remember.php

CVE-2013-1453

```
44          // Check if the highlighter is enabled.
45          if (!JComponentHelper::getParams($extension)->get('highlight_terms', 1))
46          {
47              return true;
48          }
49
50          // Check if the highlighter should be activated in this environment.
51          if (JFactory::getDocument()->getType() !== 'html' || $input->get('tmpl',
52          {
53              return true;
54          }
55
56          // Get the terms to highlight from the request.
57          $terms = $input->request->get('highlight', null, 'base64');
58          $terms = $terms ? unserialize(base64_decode($terms)) : null;
59
60          // Check the terms.
61          if (empty($terms))
62          {
63              return true;
64          }
65
66          // Clean the terms array
67          $filter = JFilterInput::getInstance();
```

CVE-2013-3242

```
31    $user = JFactory::getUser();
32    if ($user->get('guest'))
33    {
34        $hash = JApplication::getHash('JLOGIN_REMEMBER');
35
36        if ($str = JRequest::getString($hash, '', 'cookie', JREQUEST_ALLOWRAW | JREQ
37        {
38            // Create the encryption key, apply extra hardening using the user agent
39            // Since we're decoding, no UA validity check is required.
40            $privateKey = JApplication::getHash(@$_SERVER['HTTP_USER_AGENT']);
41
42            $key = new JCryptKey('simple', $privateKey, $privateKey);
43            $crypt = new JCrypt(new JCryptCipherSimple, $key);
44            $str = $crypt->decrypt($str);
45            $cookieData = @unserialize($str);
46            // Deserialized cookie could be any object structure, so make sure the
47            // credentials are well structured and only have user and password.
48            $credentials = array();
49            $filter = JFilterInput::getInstance();
50            $goodCookie = true;
```

[20130201] - Core - Information Disclosure

- **Project:** Joomla!
- **SubProject:** All
- **Severity:** Low
- **Versions:** 3.0.2 and earlier 3.0.x versions; version 2.5.8 and earlier 2.5.x versions.
- **Exploit type:** Information disclosure
- **Reported Date:** 2012-October-31
- **Fixed Date:** 2013-February-4
- **CVE Number:** CVE-2013-1453

Description

Method of encoding search terms led to possible information disclosure.

Affected Installs

Joomla! version 3.0.2 and earlier 3.0.x versions; version 2.5.8 and earlier 2.5.x versions.

Solution

Upgrade to version 3.0.3 or 2.5.9.

Reported by Egidio Romano

[20130406] - Core - DOS Vulnerability

- Project: Joomla!
- SubProject: All
- Severity: Moderate
- Versions: 2.5.9 and earlier 2.5.x versions. 3.0.3 and earlier 3.0.x versions.
- Exploit type: Denial of service vulnerability
- Reported Date: 2013-February-18
- Fixed Date: 2013-April-24
- CVE Number: CVE-2013-3242

Description

Object unserialize method leads to possible denial of service vulnerability.

Affected Installs

Joomla! version 2.5.9 and earlier 2.5.x versions; and version 3.0.2 and earlier 3.0.x versions.

Solution

Upgrade to version 2.5.10, 3.1.0 or 3.0.4.

Reported By: Egidio Romano

Exploiting Joomla

Un attaccante potrebbe sfruttare entrambe le vulnerabilità per condurre attacchi SQL Injection e Denial of Service (DoS). Ciò è possibile sfruttando il metodo distruttore della classe "plgSystemDebug", che a sua volta invoca il metodo "get" sulla proprietà "params":

```
// If the user is not allowed to view the output then end here
$filterGroups = (array) $this->params->get('filter_groups', null);
```

Exploiting Joomla: SQL Injection

Un attacco SQL Injection è possibile sfruttando il metodo "get" della classe "JCategories", che alla linea 181 invoca il metodo "_load", all'interno del quale viene eseguita una query SQL utilizzando alcune proprietà dell'oggetto:

```
166    public function get($id = 'root', $forceload = false)
167    {
168        if ($id !== 'root')
169        {
170            $id = (int) $id;
171
172            if ($id == 0)
173            {
174                $id = 'root';
175            }
176        }
177
178        // If this $id has not been processed yet, execute the _load method
179        if ((!isset($this->_nodes[$id]) && !isset($this->_checkedCategories
180        {
181            $this->_load($id);
182        }
```

Exploiting Joomla: Denial of Service

Un attacco Denial of Service è possibile sfruttando il metodo "get" della classe "JInput", che a sua volta invoca il metodo "clean" sulla proprietà "filter":

```
157    public function get($name, $default = null, $filter = 'cmd')
158    {
159        if (isset($this->data[$name]))
160        {
161            return $this->filter->clean($this->data[$name], $filter);
162        }
```

Utilizzando un oggetto di tipo "JCacheStorageFile" è possibile cancellare ricorsivamente un'intera directory, poiché il metodo "JCacheStorageFile::clean" invocherà il metodo "_deleteFolder" passandogli come parametro una stringa avente come prefisso la proprietà "_root".

Secunia
Stay Secure

Dalla teoria alla pratica...