Multiple SQL injection vulnerabilities in Joomla.
2015
Anastassia Ivanova
Tartu Ülikool
Content.

1. Introduction
2. Joomla code review
3. Statistics
4. Fixing
5. Protection
6. Summary
7. Notes
8. References
Introduction.

Joomla is a content management system (CMS [1]), which allows developers to build Web sites and powerful online applications. As any open source application, Joomla is subjected to all sorts of hacking attacks. Many attacks are occurring because of sites' extensions vulnerable to SQL injection.[2] The open-source content management framework Joomla pushed out version 3.2.3 where was fixed SQL injection zero-day vulnerability that could have let attackers steal information from databases or insert code into sites running the CMS.

As SQL databases are the heart of the Joomla system, this type of attacks is the one of the most dangerous threat current days. The database holds the content, the users' IDs, the settings, and more. Accessing the database can gain the hacker an administrative access that can gather private information such as usernames and passwords, and can allow any number of bad things to happen. With SQL injection, a malicious attack can drop the table or even a whole database in a few seconds and with a very simply command through your website. Currently, it is unacceptable for any website to have a SQL injection vulnerability, since security experts have been warning about SQL injection for at least a decade now – and, addressing SQL injection is a well-known and well-documented process.

Joomla code review.

In this section will be the plgSystemDebug::__destruct in Joomla’s code that can cause potential SQL Injection vulnerability and compare two versions of this CMS. We will take Joomla 2.5 (Figure 1.1) and 3.0 (Figure 1.2) versions for investigation.

```php
public function __destruct()
{
    // Do not render if debugging or language debug is not enabled
    if (!JDEBUG && !JFactory::getApplication()->getCfg('debug_lang'))
    {
        return;
    }
}(Figure 1.1)
```

```php
public function __destruct()
{
    // Do not render if debugging or language debug is not enabled
    if (!JDEBUG && !$this->debugLang)
    {
        return;
    }
}(Figure 1.2)
```
The System Debug plugin is enabled by default configuration, but both the Debug System and the Debug Language options are disabled. It means that in Joomla 2.5 this magic method may be abused only if one or both of those options are enabled, while in Joomla 3.0 it could always be abused because the if condition checks the debugLang object’s property. After a first sight at the source code of the plgSystemDebug::__destruct method, it might sound no useful to conduct any kind of malicious attack, but there is a line of code which calls the get method of the object stored into the params property:

```php
$filterGroups = (array) $this->params->get('filter_groups', null);
```

to call two different attack vector – Arbitrary directory deletion and Blind SQL Injection. In this thesis, we will see the last attack vector in details. At line 181 is called the JCategories::_load method, which is intended to fetch a specific category from the back-end database. Within this method is possible to inject arbitrary SQL commands through different object properties, such as _table, _field, _key and _statefield.
Statistics.

Here is a diagram describing the vulnerability level in years from 2005 up to 2012

Vulnerabilities By Year

As it can be seen, the most vulnerable period was in 2006 – 2008 years. Software can be vulnerable not only because of design or implementation errors but also because people have discovered better attacks, just as armor that offers protection from small arms fire can be penetrated by more powerful munitions. Eliminating flaws that lead to vulnerabilities is vital for IT systems, but making those that remain less potentially damaging and harder to exploit can improve security as well. Since 2008, low access - complexity vulnerabilities have dropped as a percentage of the approximately 77 percent since 2008, when the Joomla 1.5 version was released, but the proportion of high, total.

Let’s have a look at the diagram showing the vulnerability distribution by type.

Vulnerabilities By Type
According to the results above, SQL Injection is the second most popular attack in Joomla. However, this threat remains the most requiring attention because professional hacker and amateurs can do it as well. Even more SQL Injection can harm even more while it can provide private user’s data what is the aim on the SQL Injection hackers.

**Fixing.**

This vulnerability applies also to Joomla 1.5 version of the component, but the component team only fixed the vulnerability in Joomla 2.5 and 3.x versions. Here we will review how SQL injection vulnerability was fixed in version 2.5 and 3.

So, original function in Joomla 3 was:

```php
function _setExtension($option) {
    static $components = array();
    if (!isset($components[$option])) {
        $filter = ComponentUtility::getSkippedComponents();
        $component = ComponentDatabase::loadResult("SELECT `element` FROM `#__extensions` WHERE `type` = 'component' AND `element` NOT IN ({$filter}) AND `element` = '{$option}";)
    }
}
```

This has been fixed in the following way:

```php
function _setExtension($option) {
    static $components = array();
    if (!isset($components[$option])) {
        $filter = ComponentUtility::getSkippedComponents();
        $option = ComponentDatabase::escape($option);
        $component = ComponentDatabase::loadResult("SELECT `element` FROM `#__extensions` WHERE `type` = 'component' AND `element` NOT IN ({$filter}) AND `element` = '{$option}";)
    }
}
```

In Joomla 2.5, the original function was:

```php
function _setExtension($option) {
    static $components = array();
    if (!isset($components[$option])) {
        $filter = ComponentUtility::getSkippedComponents();
        $component = ComponentDatabase::loadResult("SELECT `element` FROM `#__extensions` WHERE `type` = 'component' AND `element` NOT IN ({$filter}) AND `element` = '{$option}";)
    }
}
```

This has been fixed in the following way:

```php
function _setExtension($option) {
    static $components = array();
    if (!isset($components[$option])) {
        $filter = ComponentUtility::getSkippedComponents();
    }
}
```
$option = ComponentDatabase::getEscaped($option);
$component = ComponentDatabase::loadResult("SELECT `element` FROM `#__extensions` WHERE `type` = 'component' AND `element` NOT IN (($filter)) AND `element` = '{$option}');

In Joomla 1.5, the original function is:

function _setExtension($option) {
    static $components = array();
    if (!isset($components[$option])) {
        $filter = "'com_sef', 'com_sh404sef', 'com_joomfish', 'com_config', 'com_media', 'com_installer',
            'com_templates', 'com_plugins', 'com_modules', 'com_cpanel', 'com_cache', 'com_messages', 'com_menus',
            'com_massmail', 'com_languages', 'com_users";"
        $component = ComponentDatabase::loadResult("SELECT `option` FROM `#__components` WHERE `parent` = "0" AND `option` NOT IN (".$filter.")) AND `option` = "".$option.";"
    }
    $components[$option] = $component;
}

And this has not been fixed.

So, in Joomla 3, the fixing line was:

    $option = ComponentDatabase::escape($option);

In Joomla 2.5 the fixing line was:

    $option = ComponentDatabase::getEscaped($option);

Protection.

Taking into consideration that so large CMS system, such Joomla probability being hacked is much lower (ca 80%) [3] than small CMS, this attack should be noticed. Web site security does not depend only on technical features, but on the users as well.

So how to prevent SQL injection attack on the website?

1. Keep Joomla and its extensions up-to-date

Probably the most important part of securing your Joomla website is to keep it updated to the latest version. In almost all version releases there are fixes for security issues. [4]

2. Use Strong Login Details

Usernames and passwords should not be ordinary as the hackers will easily break them. While creating a password the user should avoid password generators. Password generators use algorithms to generate the passwords which can be compromised by an attacker.
It is also recommended to use as many special characters ( *!@#$ ), numbers and capital letters in your password as possible.

3. Use Proper File Permissions & Ownership [5]

4. Use Joomla Security Extensions

jHackGuard is Joomla’s own security extension which protects the web site by filtering the data from the users’ input and implements additional PHP security settings. At the same time the plugin is disabled for the authenticated administrators so that its filters don't prevent them doing administrative tasks.

5. Backup Joomla site

Even if the user’s local machine has been compromised, the previous files and settings can be fast and easily restored.

6. Restrict the access to your admin area. [6]

The easiest way to do it is to protect the /administrator folder in the site with a strong password. Once you protect your /administrator folder an additional password will be required in order to see the standard administrator login form.

Next, you can restrict the access to the /administrator directory only to your IP address. If there isn't a file named ".htaccess" in the /administrator directory, create one and upload it via FTP for example, otherwise, just add the following lines at the end of the .htaccess file:
Summary.

Joomla is a powerful content management system, which is used by such large and famous web application as Harvard University, University of Notre Dame and Linux. This is why Joomla security must be on top level, where it actually is. Like all great IT projects, Joomla developed its security during 8-9 releases (after version 1.8 – 1.9) before becoming more or less secure. And it is still developing, because attackers also look for new methods to break the system.

According to the provided statistics and the security tips for users, it can be stated that the most effective way to protect from hackers is to not only hope for Joomla’s technical features, but do necessary steps to decrease the attack’s probability. As the SQL Injection goal is to steal as much private data as possible, it is especially important to protect Joomla’s databases. Because most attackers try to compromise these databases via user’s account, generating strong password will make the hacking process much harder and almost impossible for non-professional attackers. Nowadays there is no secure method, which can cover all possible attacks and prevent all hacking attempts. This is why it is important not to rely on developers, but try to protect our own sites in all possible way.

Notes:

[1] A content management system is software that keeps track of every piece of content on the Web site. Content can be simple text, photos, music, video, documents, or just about anything the developer can think of. A major advantage of using a CMS is that it requires almost no technical skill or knowledge to manage.

[2] This a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution. For example, suppose we have the following server code:

```
uName = getRequestString("UserName");
uParam = getRequestString("UserPass");

sql = "SELECT * FROM Users WHERE Name =" + uName + " AND Pass =" + uParam + ""
```

A smart hacker might get access to user names and passwords in a database by simply inserting " or "]=" into the user name or password text box.

The code at the server will create a valid SQL statement like this:

```
SELECT * FROM Users WHERE Name =" or "]=" AND Pass =" or "]="
```

The result SQL is valid. It will return all rows from the table Users, since WHERE "]=" is always true.
Here is a diagram showing the probability to be attacked by SQL Injection

Web applications most vulnerable to XSS and SQL injection attacks

- In-house web applications: 40%
- Plugins and modules for CMSs: 30%
- Small CMSs: 25%
- Large CMSs (WordPress, Joomla, etc.): 5%


[5] Change file permissions

[6] How to protect folders

References:

http://www.joomla.org/about-joomla.html
https://www.siteground.com/tutorials/joomla/joomla-security.htm
https://www.packtpub.com/books/content/preventing-sql-injection-attacks-your-joomla-websites