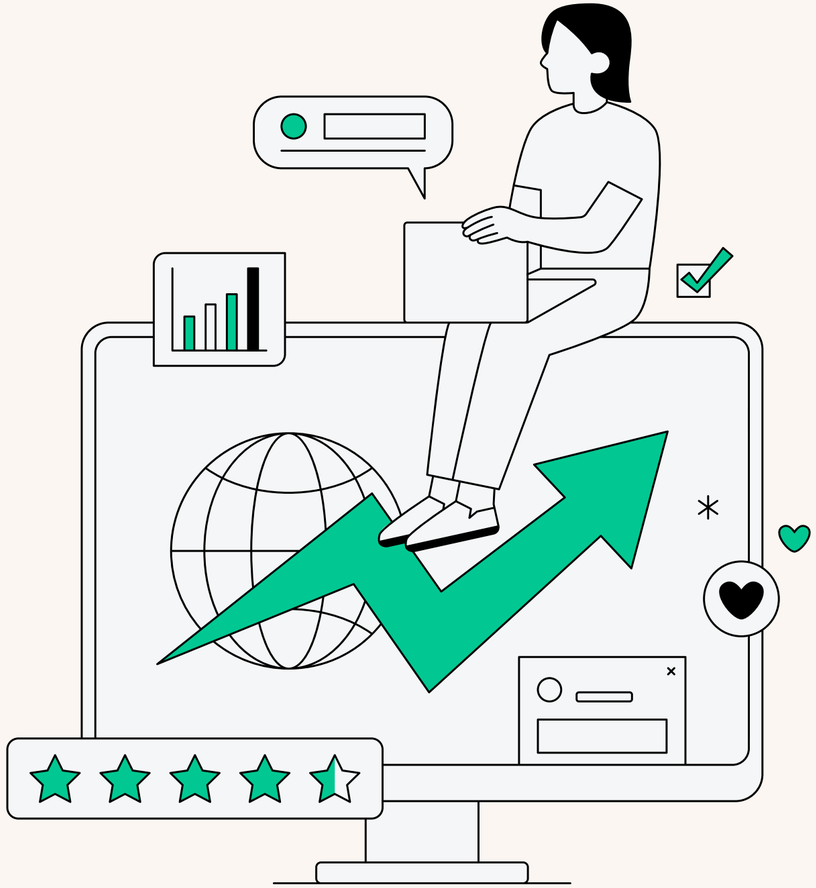
Presented by Monu Dhakad

Vulnerability Report

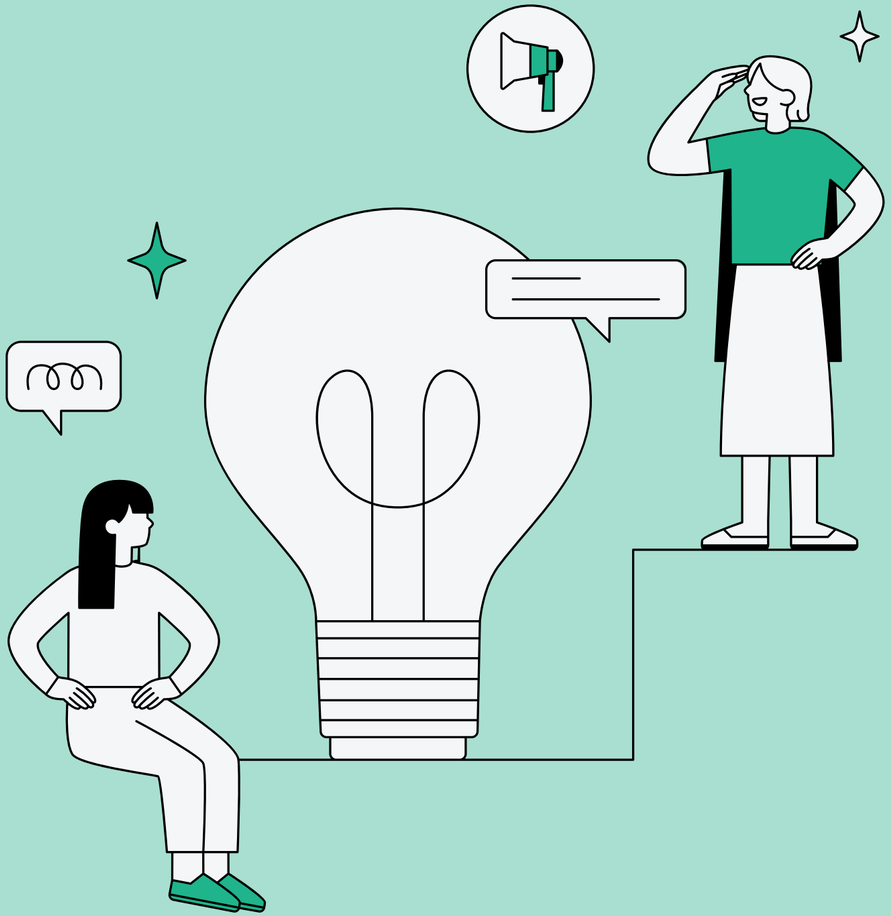
Internship studio project

http://testasp.vulnweb.com/



Introduction

As per the task's description, this slide will contain a screenshot of Invicti-found vulnerabilities (Note that Netsparker is now renamed as Invicti), screenshots of the report that Invicti generated, and a report that I've created so that you can check the difference between the actual and my submitted report



invicti

12/19/2023 11:18:35 AM (UTC+05:30)

Detailed Scan Report

http://testasp.vulnweb.com/

Scan Time : 12/19/2023 8:59:04 AM (UTC+05:30)
Scan Duration : 00:01:10:49

Total Requests: 13,940
Average Speed: 3.3r/s

Risk Level:
CRITICAL

VULNERABILITIES

28	15	4	2	2	10	3	7
IDENTIFIED	CONFIRMED	CRITICAL	HIGH	MEDIUM	LOW	BEST PRACTICE	INFORMATION

Identified Vulnerabilities

Critical	4
High	2
Medium	2
Low	10
Best Practice	3
Information	7
TOTAL	**28**

Confirmed Vulnerabilities

Critical	4
High	2
Medium	0
Low	4
Best Practice	1
Information	4
TOTAL	**15**

Vulnerability Summary

SEVERITY FILTER : ☑ CRITICAL ☐ HIGH ☐ MEDIUM ☐ LOW ☐ BEST PRACTICE ☐ INFORMATION

CONFIRM	VULNERABILITY	METHOD	URL	PARAMETER	PARAMETER TYPES
	Boolean Based SQL Injection	POST	http://testasp.vulnweb.com/Login.asp?RetURL=%2FDefault.asp%3F	tfUPass	Post
	Boolean Based SQL Injection	POST	http://testasp.vulnweb.com/Login.asp?RetURL=%2FDefault.asp%3F	tfUName	Post
	Boolean Based SQL Injection	GET	http://testasp.vulnweb.com/showforum.asp?id=0%20OR%2017-7%3d10	id	Querystring
	Boolean Based SQL Injection	GET	http://testasp.vulnweb.com/showthread.asp?id=0%20OR%2017-7%3d10	id	Querystring

Boolean Based SQL Injection

CONFIRMED CRITICAL

URL : http://testasp.vulnweb.com/showforum.asp?id=0%20OR%2017-7=10
Parameter Name : id
Parameter Type : GET
Attack Pattern : 0+OR+17-7%3d10

Vulnerability details

Invicti Standard identified a Boolean-Based SQL Injection, which occurs when data input by a user is interpreted as a SQL command rather than as normal data by the backend database.

This is an extremely common vulnerability and its successful exploitation can have critical implications.

Invicti Standard confirmed the vulnerability by executing a test SQL query on the backend database. In these tests, SQL injection was not obvious, but the different responses from the page based on the injection test allowed Invicti Standard to identify and confirm the SQL injection.

Impact

Depending on the backend database, the database connection settings, and the operating system, an attacker can mount one or more of the following type of attacks successfully:

- Reading, updating and deleting arbitrary data/tables from the database
- Executing commands on the underlying operating system

Actions to Take

- See the remedy for solution
- If you are not using a database access layer (DAL), consider using one. This will help you centralize the issue. You can also use ORM (object relational mapping). Most of the ORM systems use only parameterized queries and this can solve the whole SQL injection problem
- Locate all of the dynamically generated SQL queries and convert them to parameterized queries. (If you decide to use a DAL/ORM, change all legacy code to use these new libraries.)
- Use your weblogs and application logs to see if there were any previous but undetected attacks to this resource

Remedy

The best way to protect your code against SQL injections is using parameterized queries (prepared statements). Almost all modern languages provide built-in libraries for this. Wherever possible, do not create dynamic SQL queries or SQL queries with string concatenation.

Required Skills for Successful Exploitation

There are numerous freely available tools to exploit SQL injection vulnerabilities. This is a complex area with many dependencies; however, it should be noted that the numerous resources available in this area have raised both attacker awareness of the issues and their ability to discover and leverage them.

External References

- OWASP SQL injection
- SQL Injection Cheat Sheet
- SQL Injection Vulnerability

Remedy References

- A guide to preventing SQL injection
- SQL injection Prevention Cheat Sheet

My Report

summary:

The website has an endpoint that is vulnerable to an injection vulnerability named - Boolean-Based SQL injection (a type of SQL Injection), where the attacker sends SQL queries to the database, forcing the application to return a different result depending on whether the query returns a true or false result.
This is a very common and critical vulnerability that can cause a very serious impact.

So, I ran this test where I tried a SQL query on the database behind the scenes. At first, I didn't see any obvious SQL injection, but the way the page responded during my test helped me spot and confirm the SQL injection vulnerability.

Required Skills for Successful Exploitation

tons of free tools floating around help people exploit these weaknesses in a system. It's not an easy deal, however. There's a lot to it, nut any attackers know how to deal with them

the thing is that several tools have made it easier for attackers to get a grip on these problems They're more aware of the issues now, and they've gotten better at spotting and using these vulnerabilities.

the more these tools are out there, the more the bad guys know about these gaps, and the more they can exploit them.

Impact

depending on what kind of database is in the background, how it's connected, and what operating system is running, any person could do a few different types of attacks that might work:

- They could read, change, or even delete any data or tables they want from the database.
- They might even be able to run commands on the actual operating system that's running everything. Like, they could make the system do things it shouldn't, just by exploiting these weaknesses.

Actions to Take

- if you're not already using a database access layer (DAL), you should get one. It's like a way to centralize stuff and keep things in line.
- Another cool thing to check out is ORM (object-relational mapping). With ORM, it's mostly about using parameterized queries, which can fix up the whole SQL injection mess. It's like a built-in way to keep things safe and sorted when you're working with databases.
- Find all those SQL queries that are dynamically generated, and switch those up to use parameterized queries instead. If you decide to go with a database access layer (DAL) or an object-relational mapping (ORM) thing, make sure to update all the old code to use these new libraries. It's like giving your system an upgrade to keep things secure.
- Check out your web and app logs to see if there were any attacks on this resource before that went unnoticed. Sometimes, those logs hold clues about stuff that happened but wasn't picked up at the time.

Vulnerabilities

- http://testasp.vulnweb.com/showforum.asp?id=0%20OR%2017-7%3d10

Request | Response

Response Time (ms) : 291.6664 Total Bytes Received : 3979 Body Length : 3802 Is Compressed : No

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Content-Length: 3802
Content-Type: text/html
Date: Tue, 19 Dec 2023 03:33:27 GMT
Cache-Control: private


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><!-- InstanceBegin template="/Templates/MainTemplate.dwt.asp" codeOutsideHTMLIsLocked="false" -->
<head>
<!-- InstanceBeginEditable name="doctitle" -->
<title>acuforum Acunetix Web Vulnerability Scanner</title>
<!-- InstanceEndEditable -->
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- InstanceBeginEditable name="head" -->
<!-- InstanceEndEditable -->
<link href="styles.css" rel="stylesheet" type="text/css">
</head>
<body>
<table width="100%" border="0" cellpadding="10" cellspacing="0">
<tr bgcolor="#008F00">
<td width="306px"><a href="https://www.acunetix.com/"><img src="Images/logo.gif" width="306" height="38" border="0" alt="Acunetix website security"></a></td>
<td align="right" valign="middle" bgcolor="#008F00" class="disclaimer">TEST and Demonstration site for <a href="https://www.acunetix.com/vulnerability-scanner/">Acunetix Web Vulnerability Scanner</a></td>
</tr>
<tr>
<td colspan="2"><div class="menubar"><a href="Templatize.asp?item=html/about.html" class="menu">about</a> - <a href="Default.asp" class="menu">forums</a> - <a href="Search.asp" class="menu">search</a>
- <a href="./Login.asp?RetURL=%2Fshowforum%2Easp%3Fid%3D0%2520OR%252017%2D7%253d10" class="menu">login</a> - <a href="./Register.asp?RetURL=%2Fshowforum%2Easp%3Fid%3D0%2520OR%252017%2D7%253d10" class="menu">register</a>
- <a href="https://www.acunetix.com/vulnerability-scanner/sql-injection/" class="menu">SQL scanner</a> - <a href="https://www.acunetix.com/websitesecurity/sql-injection/" class="menu">SQL vuln help</a>
</div></td>
</tr>
<tr>
<td colspan="2"><!-- InstanceBeginEditable name="MainContentLeft" -->
<
_
```

Remedy

To keep your code safe from SQL injections, the go-to move is using parameterized queries, also known as prepared statements. Most new languages have their own ready-made libraries for this security trick. Try to avoid making dynamic SQL queries or joining strings to form SQL queries whenever you can.

External References

- OWASP SQL injection
- SQL Injection Cheat Sheet
- SQL Injection Vulnerability

Remedy References

- A guide to preventing SQL injection
- SQL injection Prevention Cheat Sheet

Presented by Monu Dhakad

Thank you very much!

monudhakad2332005@gmail.com