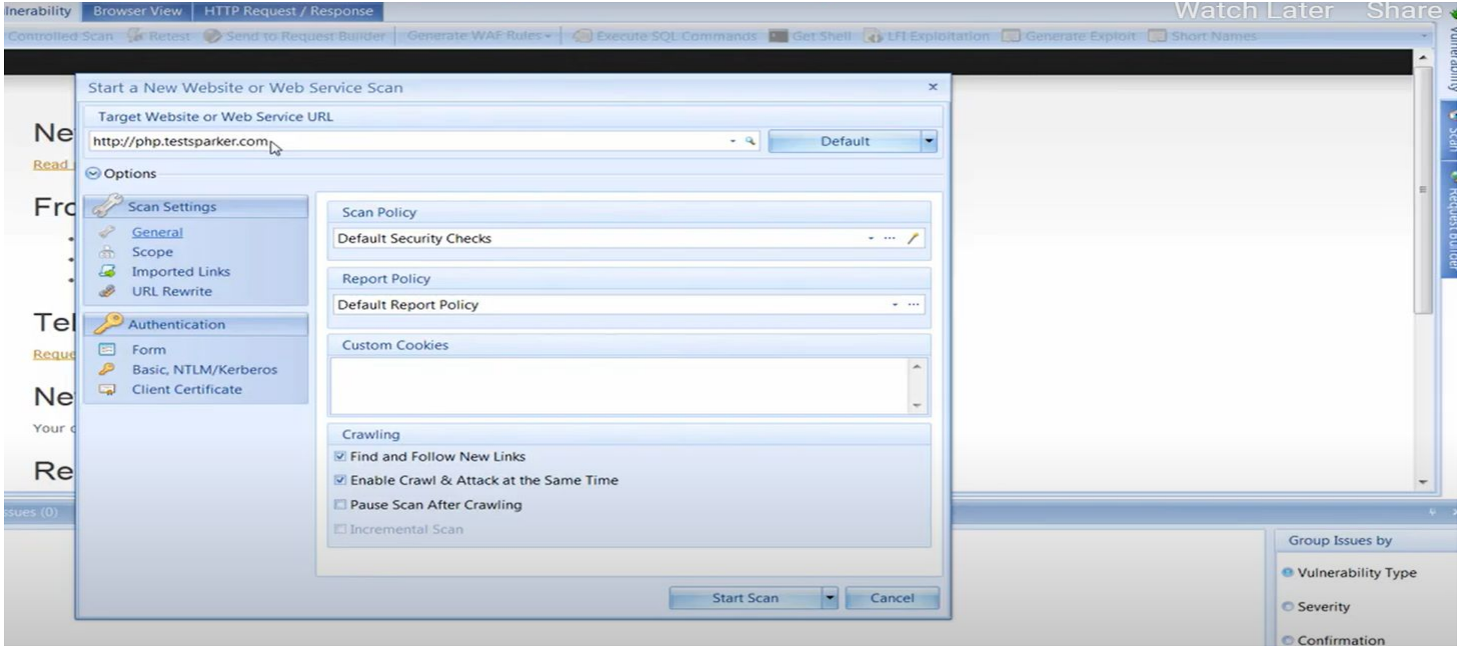
VULNERABILITIES USING NETSPARKER

TARGET WEBSITE: http://php.testsparker.com

Initial screen



Start a New Website or Web Service Scan

Target Website or Web Service URL

http://php.testsparker.com

Default

Scan Policy

Default Security Checks

Report Policy

Default Report Policy

Custom Cookies

Crawling

☑ Find and Follow New Links

☑ Enable Crawl & Attack at the Same Time

☐ Pause Scan After Crawling

☐ Incremental Scan

Group Issues by

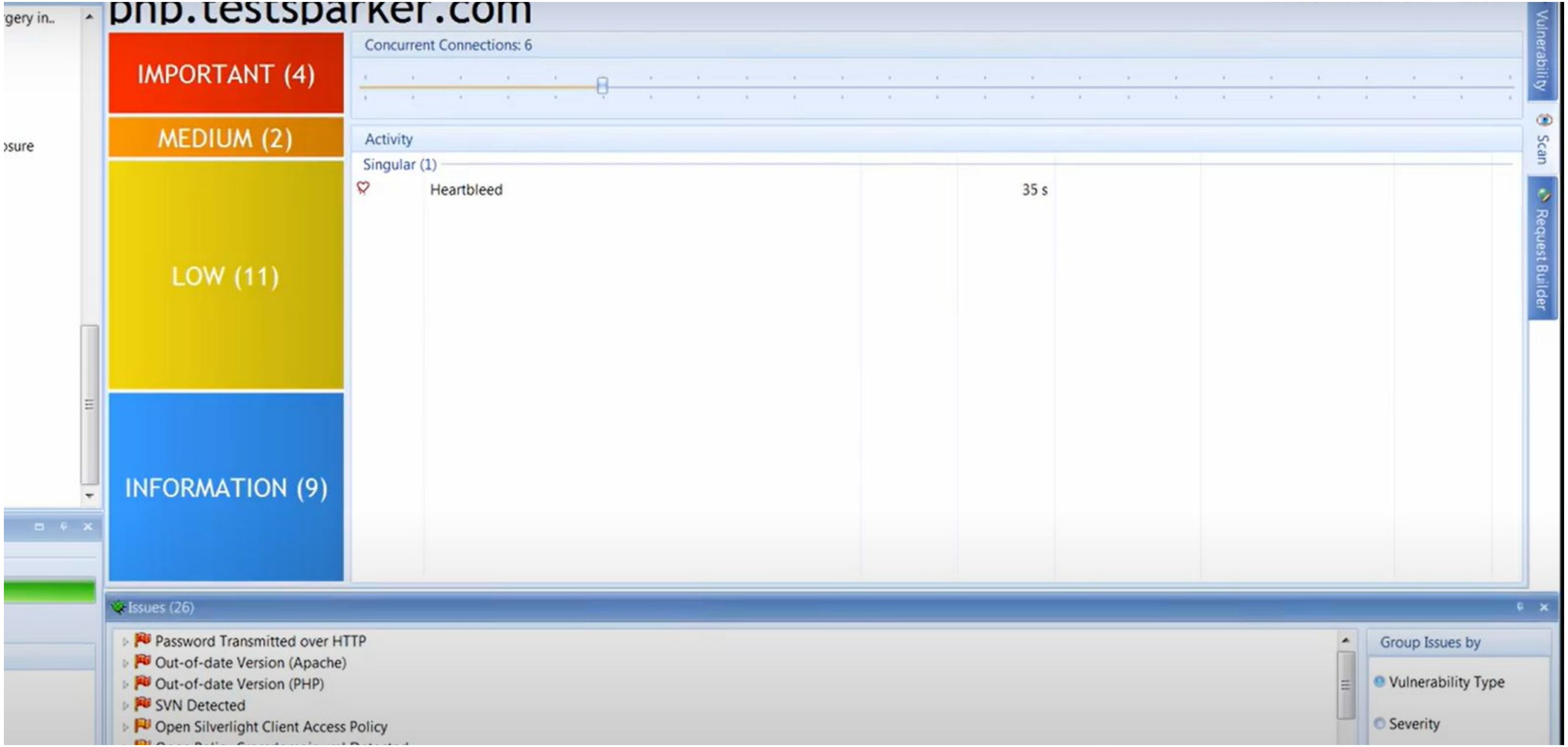
● Vulnerability Type

○ Severity

○ Confirmation

While using netsparker

‘



php.testsparker.com

Concurrent Connections: 6

Activity

Singular (1)

Heartbleed 35 s

Issues (26)

- Password Transmitted over HTTP
- Out-of-date Version (Apache)
- Out-of-date Version (PHP)
- SVN Detected
- Open Silverlight Client Access Policy

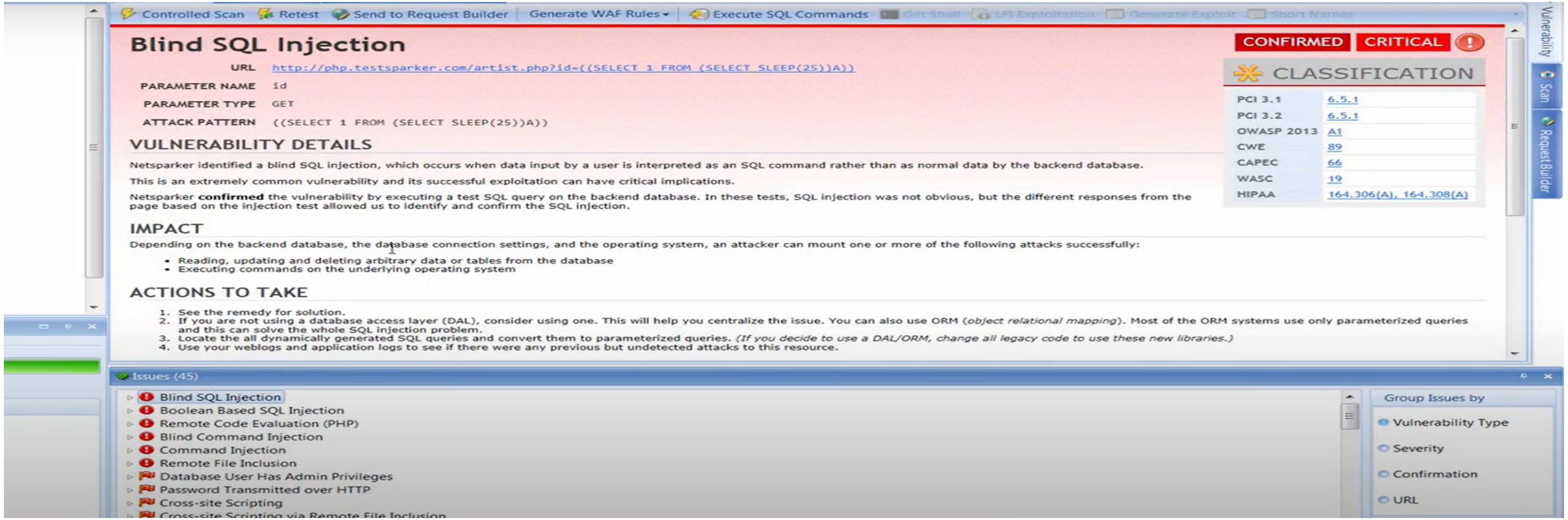
Group Issues by

● Vulnerability Type

○ Severity

Vulnerabilities found

There were many vulnerabilities like: blind sql injection,boolean based sql injection,password transmitted over HTTP



Blind SQL Injection

URL	http://php.testsparker.com/artist.php?id=((SELECT 1 FROM (SELECT SLEEP(25))A))
PARAMETER NAME	id
PARAMETER TYPE	GET
ATTACK PATTERN	((SELECT 1 FROM (SELECT SLEEP(25))A))

CLASSIFICATION

PCI 3.1	6.5.1
PCI 3.2	6.5.1
OWASP 2013	A1
CWE	89
CAPEC	66
WASC	19
HIPAA	164.306(A), 164.308(A)

VULNERABILITY DETAILS

Netsparker identified a blind SQL injection, which occurs when data input by a user is interpreted as an 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.

Netsparker **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 us 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 attacks successfully:

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

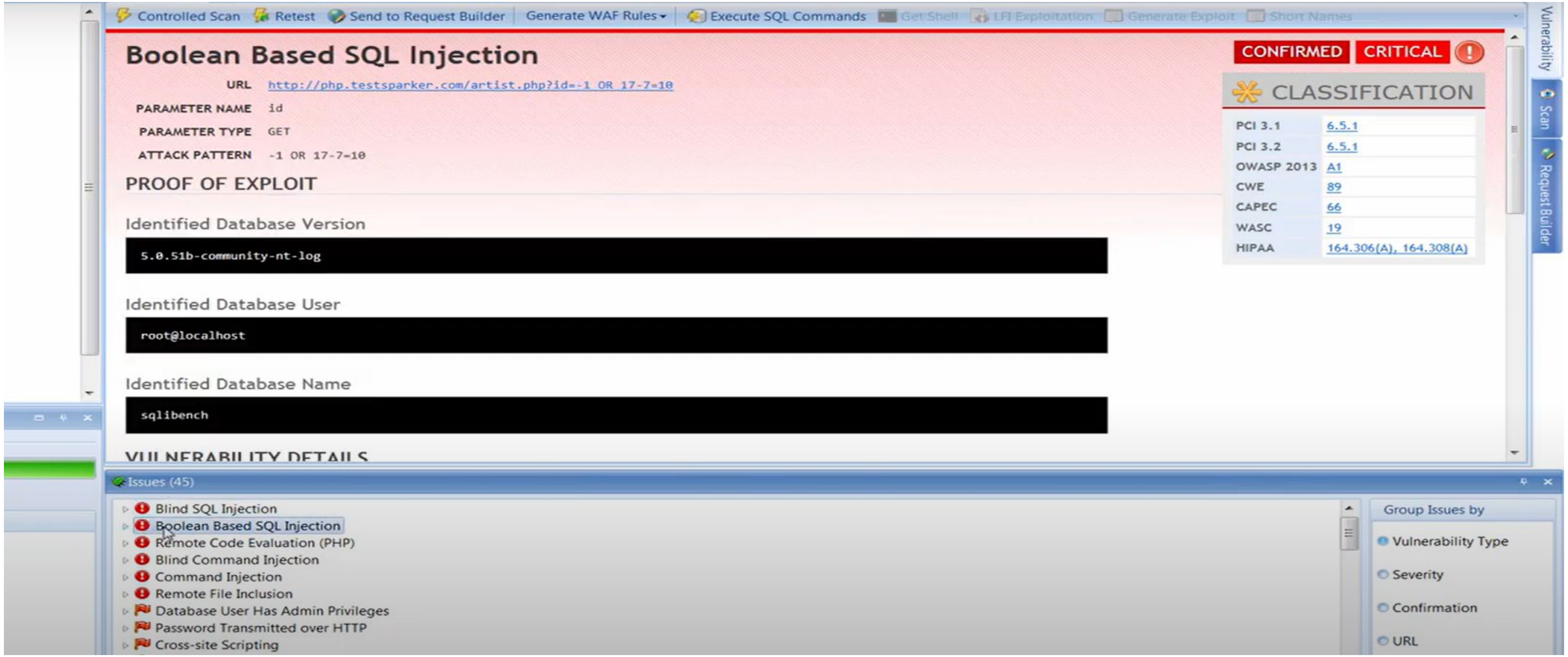
ACTIONS TO TAKE

1. See the remedy for solution.
2. 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.
3. Locate the all 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.)*
4. Use your weblogs and application logs to see if there were any previous but undetected attacks to this resource.

Issues (45)

- Blind SQL Injection
- Boolean Based SQL Injection
- Remote Code Evaluation (PHP)
- Blind Command Injection
- Command Injection
- Remote File Inclusion
- Database User Has Admin Privileges
- Password Transmitted over HTTP
- Cross-site Scripting
- Cross-site Scripting via Remote File Inclusion

vulnerability(2)



Boolean Based SQL Injection

CONFIRMED CRITICAL

URL	http://php.testsparker.com/artist.php?id=-1 OR 17-7=10
PARAMETER NAME	id
PARAMETER TYPE	GET
ATTACK PATTERN	-1 OR 17-7=10

CLASSIFICATION

PCI 3.1	6.5.1
PCI 3.2	6.5.1
OWASP 2013	A1
CWE	89
CAPEC	66
WASC	19
HIPAA	164.306(A), 164.308(A)

PROOF OF EXPLOIT

Identified Database Version

```
5.0.51b-community-nt-log
```

Identified Database User

```
root@localhost
```

Identified Database Name

```
sqlibench
```

VULNERABILITY DETAILS

Issues (45)

- Blind SQL Injection
- Boolean Based SQL Injection
- Remote Code Evaluation (PHP)
- Blind Command Injection
- Command Injection
- Remote File Inclusion
- Database User Has Admin Privileges
- Password Transmitted over HTTP
- Cross-site Scripting

Group Issues by

● Vulnerability Type

○ Severity

○ Confirmation

○ URL

BLIND SQL INJECTION

WHAT IS BLIND SQL INJECTION?

Blind SQL injection arises when an application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

With blind SQL injection vulnerabilities, many techniques such as `UNION` attacks, are not effective because they rely on being able to see the results of the injected query within the application's responses. It is still possible to exploit blind SQL injection to access unauthorized data, but different techniques must be used.

What happens in sql injection attack?

After the attacker verifies the presence of an SQL Injection vulnerability, they can try different requests (often involving UNION SELECT statements) to receive information about the database in error responses. They can use it to fingerprint the database (find out if it’s MySQL, PostgreSQL, Oracle, MSSQL, etc. and which version), build the database schema, retrieve data from any table in the database, and escalate the attack.

Web server administrators quickly realized that showing errors to the general public is not a wise thing to do, so they started suppressing detailed error messages. This is a flawed solution because it does not address the underlying problem. The SQL interpreter can still parse user input as part of an SQL query.

Attackers came up with methods to go around the lack of error messages and still know if the input is being interpreted as an SQL statement. This is how the *Blind SQL Injection* technique was born (sometimes called *Inferential SQL Injection*). There are two variants of this technique that are commonly used: *Content-based Blind SQL Injection* and *Time-based Blind SQL Injection*.

HOW TO PREVENT SQL INJECTION ATTACK

Although the techniques needed to find and exploit blind SQL injection vulnerabilities are different and more sophisticated than for regular SQL injection, the measures needed to prevent SQL injection are the same regardless of whether the vulnerability is blind or not.

As with regular SQL injection, blind SQL injection attacks can be prevented through the careful use of parameterized queries, which ensure that user input cannot interfere with the structure of the intended SQL query.