

Systems Security & Project



L2 - Secure Coding Techniques

Content

- Intro
- Secure software development
- Good coding practices
- Principles of secure code design
- Example

Secure Coding Techniques

Introduction

**Security
Guidelines**

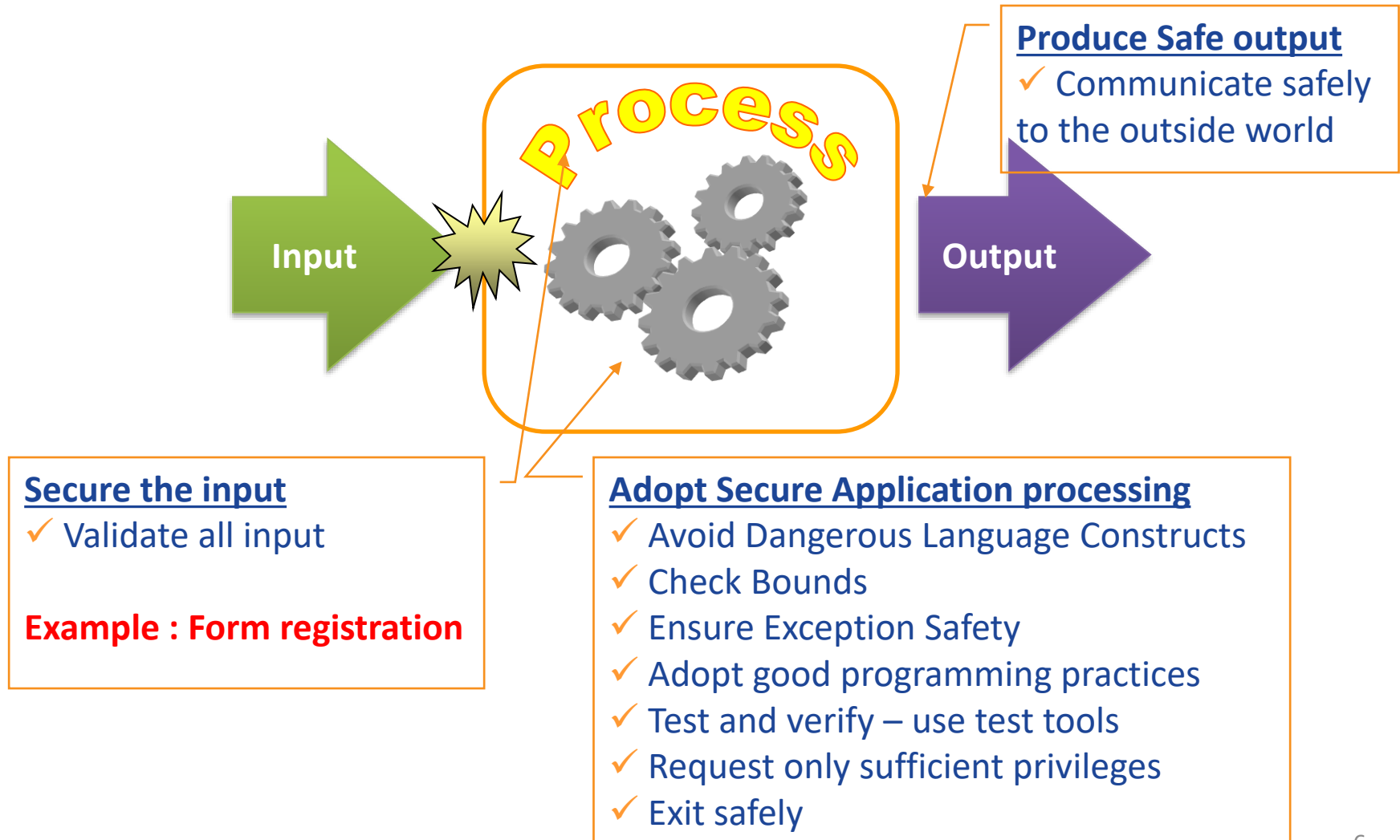
**Security
Strategies**

**Security
Principles**

How to build Secure Software?

- It is IMPOSSIBLE to build a perfect software.
- It is impractical to eliminate all security vulnerabilities.
- However, we can do our BEST in building software that is both secure & reliable.
- Developers need to be aware of current software security issues.
- Developers and project managers can use automated solutions to check software code to ensure security and reliability.

Basic Application Security Guidelines



Secure Coding Techniques

Introduction

**Security
Guidelines**

**Security
Strategies**

**Security
Principles**

Security Guidelines

- There are many good coding practices available from Microsoft, Oracle, and other large and small software providers on the Internet that provide excellent direction on securing your code.

Security Requirements Build from Published Best Practices & Vulnerabilities Info

- [The Open Web Application Security Project \(OWASP\)](#)
- [SANS - The Top Cyber Security Risks](#)
- [SANS Top 25 Programming Errors](#)
- [Qualys - Top 10](#)
- [MITRE ATT&CK®](#)
- Common Weakness Enumeration (CWE)
 - <http://cwe.mitre.org/>
 - <http://cwe.mitre.org/data/graphs/699.html>
- [Common Vulnerabilities and Exposures \(CVE®\)](#)

Security Guidelines

In this module, we explore three approaches to encourage secure coding practices:
CWE/SAN, MITRE and OWASP:

1. CWE/SANS Top 25 software errors
2. OWASP Secure coding practices checklist
3. MITRE ATT&CK

01 CWE/SANS Top 25 Software Errors

- The SANS Institute has a global reputation for been neutral on products, vendors, and standards.
- The CWE/SANS Top 25 software errors explore the fundamental failures in programming that promote security weaknesses and criminal exploitation. *CWE List Version 4.5*

CWE

Common Weakness Enumeration

- They help developers and security practitioners to:
 - Describe and discuss software and hardware weaknesses in a common language.
 - Check for weaknesses in existing software and hardware products.
 - Evaluate coverage of tools targeting these weaknesses.
 - Leverage a common baseline standard for weakness identification, mitigation, and prevention efforts.
 - Prevent software and hardware vulnerabilities prior to deployment.

01 CWE/SANS Top 25 Software Errors

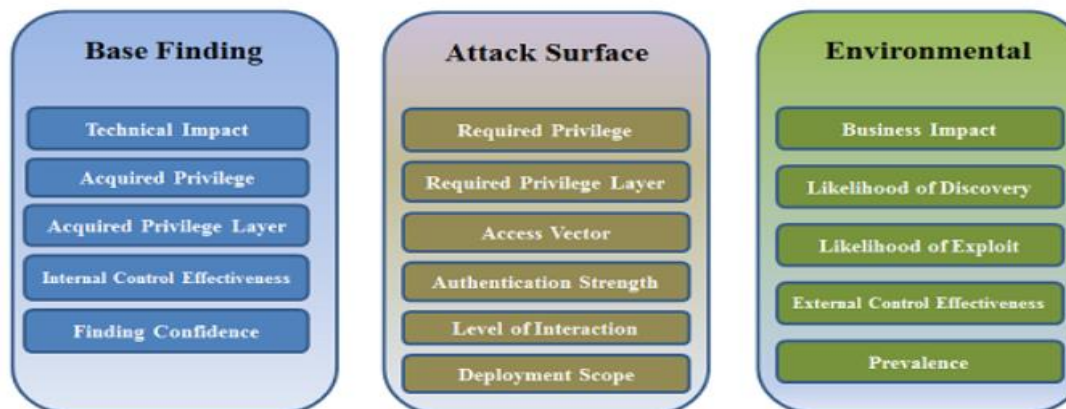
Common Weakness Scoring System (CWSS™)

- CWSS provides a mechanism for prioritizing software weaknesses in a consistent, flexible, open manner.
- It is a collaborative, community-based effort that is addressing the needs of its stakeholders across government, academia, and industry.

01 CWE/SANS Top 25 Software Errors

Common Weakness Scoring System (CWSS™)

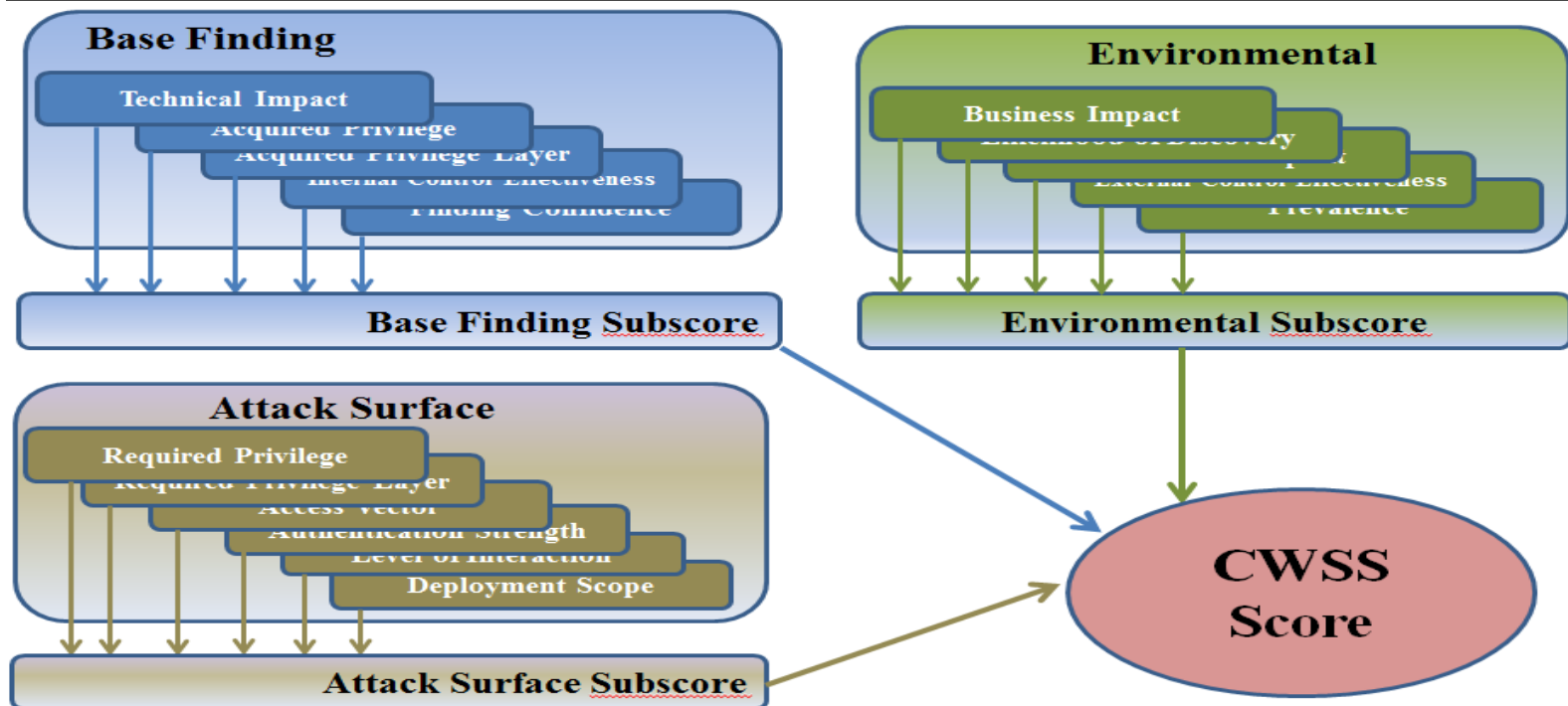
- CWSS is organized into three metric groups:
 - Base Finding, Attack Surface, and Environmental.
 - Each group contains multiple metrics - also known as factors - that are used to compute a CWSS score
- **Base Finding metric group**: captures the inherent risk of the weakness, confidence in the accuracy of the finding, and strength of controls.
- **Attack Surface metric group**: the barriers that an attacker must overcome in order to exploit the weakness.
- **Environmental metric group**: characteristics of the weakness that are specific to a particular environment or operational context.



01 CWE/SANS Top 25 Software Errors

Common Weakness Scoring System (CWSS™)

- three subscores are multiplied together, which produces a CWSS score between 0 and 100.



01 CWE/SANS Top 25 Software Errors

Base finding matrix group

- The Base Finding metric group consists of the following factors:
 - **Technical Impact (TI)** (See table below)
 - Acquired Privilege (AP)
 - Acquired Privilege Layer (AL)
 - Internal Control Effectiveness (IC)
 - Finding Confidence (FC)

Read section of 2.3 of https://cwe.mitre.org/cwss/cwss_v1.0.1.html

Value	Code	Weight	Description
Critical	C	1.0	Complete control over the software being analyzed, to the point where operations cannot take place.
High	H	0.9	Significant control over the software being analyzed, or access to critical information can be obtained.
Medium	M	0.6	Moderate control over the software being analyzed, or access to moderately important information can be obtained.
Low	L	0.3	Minimal control over the software being analyzed, or only access to relatively unimportant information can be obtained.
None	N	0.0	There is no technical impact to the software being analyzed at all. In other words, this does not lead to a vulnerability.
Default	D	0.6	The Default weight is the median of the weights for Critical, High, Medium, Low, and None.
Unknown	UK	0.5	There is not enough information to provide a value for this factor. Further analysis may be necessary. In the future, a different value might be chosen, which could affect the score.
Not Applicable	NA	1.0	This factor is being intentionally ignored in the score calculation because it is not relevant to how the scorer prioritizes weaknesses. This factor might not be applicable in an environment with high assurance requirements; the user might want to investigate every weakness finding of interest, regardless of confidence.
Quantified	Q		This factor could be quantified with custom weights.

01 CWE/SANS Top 25 Software Errors

https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5
[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1
[13]	CWE-502	Deserialization of Untrusted Data	6.71	+8
[14]	CWE-287	Improper Authentication	6.58	0
[15]	CWE-476	NULL Pointer Dereference	6.54	-2
[16]	CWE-798	Use of Hard-coded Credentials	6.27	+4
[17]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	CWE-862	Missing Authorization	5.47	+7
[19]	CWE-276	Incorrect Default Permissions	5.09	+22
[20]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
[21]	CWE-522	Insufficiently Protected Credentials	4.21	-3
[22]	CWE-732	Incorrect Permission Assignment for Critical Resource	4.2	-6
[23]	CWE-611	Improper Restriction of XML External Entity Reference	4.02	-4
[24]	CWE-918	Server-Side Request Forgery (SSRF)	3.78	+3
[25]	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

01 CWE/SANS Top 25 Software Errors

Rank	ID	Name	2020 Rank Change
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	-1
[4]	CWE-20	Improper Input Validation	-1
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	0
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	+5
[11]	CWE-306	Missing Authentication for Critical Function	+13
[13]	CWE-502	Deserialization of Untrusted Data	+8
[14]	CWE-287	Improper Authentication	0
[16]	CWE-798	Use of Hard-coded Credentials	+4
[18]	CWE-862	Missing Authorization	+7
[19]	CWE-276	Incorrect Default Permissions	+22
[20]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	-13
[21]	CWE-522	Insufficiently Protected Credentials	-3
[23]	CWE-611	Improper Restriction of XML External Entity Reference	-4
[24]	CWE-918	Server-Side Request Forgery (SSRF)	+3

01 CWE/SANS Top 25 Software Errors



Home > CWE List > CWE- Individual Dictionary Definition (4.0)

Home

About

CWE List

Scoring

Community

CWE-20: Improper Input Validation

Weakness ID: 20

Abstraction: Class

Structure: Simple

[CWE-20 : improper Input Validation](#)

Presentation Filter:

▼ Description

The product does not validate or incorrectly validates input that can affect the control flow or data flow c

▼ Extended Description

When software does not validate input properly, an attacker is able to craft the input in a form that is no
lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrar

▼ Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. TI
MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In a

01 CWE/SANS Top 25 Software Errors

Improper Input Validation

▼ Common Consequences

Scope	Impact
Availability	Technical Impact: DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory) An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.
Confidentiality	Technical Impact: Read Memory; Read Files or Directories An attacker could read confidential data if they are able to control resource references.
Integrity Confidentiality Availability	Technical Impact: Modify Memory; Execute Unauthorized Code or Commands An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.

▼ Likelihood Of Exploit

High

01 CWE/SANS Top 25 Software Errors

Improper Input Validation

- When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application.
- This will lead to
 - unintended input resulting in altered control flow or code execution.
 - SQL injection attack
 - Buffer overflow attack etc..
- Example :
 - Upload a file that has no .pdf or .zip extension
 - Upload a file larger than 100 kB
 - Accepting special chars !@\$ as userID during registration etc ..

Input validation

- Also known as data validation
 - proper testing of any input supplied by a user or application.
- What can be validated ?
 - password, email, date, mobile numbers and more fields.
- Input validation prevents improperly formed data from entering an information system.
 - Because it is difficult to detect a malicious user who is trying to attack software.

Validate All Inputs

- File content e.g only allow .pdf, .gif, .jpg
- All input from all sources must be carefully validated
 - DO NOT just check for illegal input
- Limit maximum input character length
- Numbers: check bounds - min & max
- Make sure encodings (e.g. UTF-8, URL encoding) are legal & decoded results are legal
- Aware of various data types & input sources
 - Watch out for special characters (e.g. ' , < , >)

Input Validation

There are two main characteristics of validation that differ between implementations:

- Classification strategy (Input Filtering)
 - User input can be classified using either blacklisting or whitelisting.
- Validation outcome
 - User input identified as malicious can either be rejected or sanitised.

Improper Input Validation (Prevention)

Input Filtering

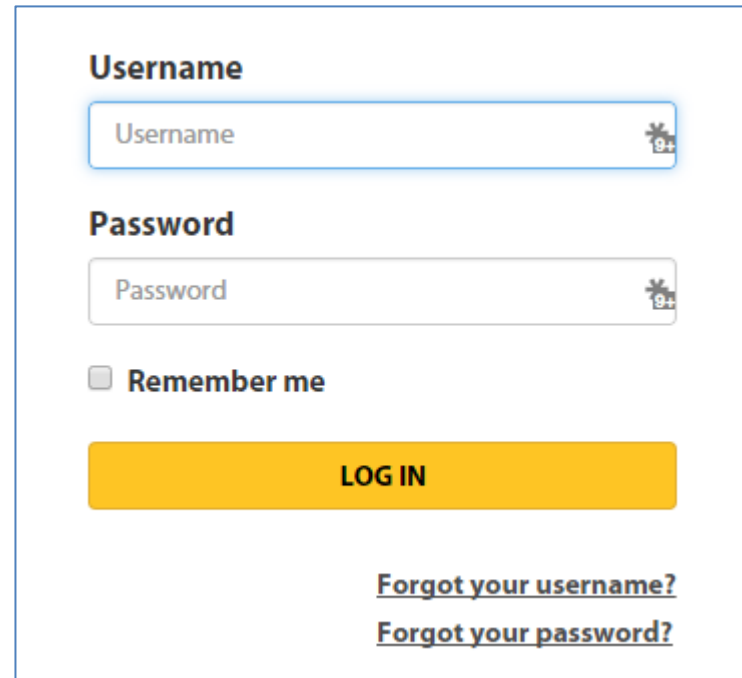
A decision making process that leads either to the acceptance or the rejection of input based on predefined criteria.

- Acceptable input will be processed and unwanted input will be blocked.
 - There are two major approaches to input filtering:
 - **Whitelist** - Allowing only the known good characters. E.g. a-z,A-Z,0-9 are known good characters in the whitelist and are hence accepted by the filter
 - **Blacklist** - Allowing anything except the known bad characters. E.g. <,/,> are known bad characters in the blacklist and are hence blocked by the filter

Improper Input Validation (Prevention)

Input Filtering

- Whitelist/blacklist for username ?
- Whitelist/blacklist for password?



The image shows a login form with the following elements:

- Username**: A text input field with the placeholder text "Username".
- Password**: A text input field with the placeholder text "Password".
- ☐ **Remember me**: A checkbox with the label "Remember me".
- LOG IN**: A yellow button with the text "LOG IN".
- [Forgot your username?](#): A link for users who forgot their username.
- [Forgot your password?](#): A link for users who forgot their password.

Password Policy

Is this a good password policy?

A password policy is a set of rules designed to increase the security of your 4me account by encouraging users to create and use strong passwords.

The existing password policy can be modified below. This password policy applies to all people who are registered in this account.

Minimum password length

- ☒ Require at least one uppercase letter
- ☒ Require at least one lowercase letter
- ☒ Require at least one number
- ☒ Require at least one symbol character

Password expires in days

Enforce password history passwords remembered

Save

Input Filtering

Blacklisting - Allowing everything except ...

- perform validation by defining a forbidden pattern that should not appear in user input.
- If a string matches this pattern, it is then marked as invalid.
 - An example would be to allow users to submit custom URLs with any protocol except javascript : e.g *javascript:alert (...) ;*

Blacklisting – drawback

Complexity

- Accurately describing the set of all possible malicious strings is **usually a very complex task**.
- The example described above could not be successfully implemented by simply searching for the substring "javascript"
 - would miss strings of the form "Javascript:" (where the first letter is capitalized) and
 - "javascript:" (where the first letter is encoded as a numeric character reference). [Ascii](#) where 106 represent 'j'

Blacklisting – drawback

Staleness

- a new features not included in the blacklist.
 - For example, an HTML validation blacklist developed before the introduction of of some attribute would fail to stop an attacker.
- web development is made up of many different technologies that are constantly being updated.

Input Filtering

Whitelisting is essentially the opposite of blacklisting:

- instead of defining a forbidden pattern, a whitelist approach defines an allowed pattern and marks input as invalid if it does not match this pattern.
 - Example :validate usernames that allows only lowercase, numerals and min 3 and max 16 chars
 - Use regular expression : **^[a-z0-9_]{3,16}\$**

Input Filtering

Whitelisting vs Blacklisting

- When building secure software, whitelisting is the recommended minimal approach.
- Blacklisting is prone to error and can be bypassed with various evasion techniques. However, it can be useful to help detect obvious attacks. (e.g keywords like `<script>` in the input textbox)
- *Usage*
 - *whitelisting* helps limit the attack surface by ensuring data is of the right semantic validity.
 - *blacklisting* helps detect and potentially stop obvious attacks.

End of Section 1

Next Session ...
Regular Expression

Regular expression

Regular expressions offer a way to check
whether data matches a specific pattern
(Whitelist)

Flexible matching

- Within regex there are many characters with special meanings – metacharacters

- star (*) matches any number of instances

`/ab*c/` => 'a' followed by **zero or more** 'b' followed by 'c'

- plus (+) matches at least one instance

`/ab+c/` => 'a' followed by **1 or more** 'b' followed by 'c'

- question mark (?) matches zero or one instance

`/ab?c/` => 'a' followed by **0 or 1** 'b' followed by 'c'

More Flexibility

- Match a character a specific number or range of instances
- **{x}** will match x number of instances.
`/ab{3}c/ => abbbc`
- **{x,y}** will match between x and y instances.
`/a{2,4}bc/ => aabc or aaabc or aaaabc`
- **{x, }** will match x+ instances.
`/abc{3, }/ => abccc or abcccccc or abccccccccc`

More Flexibility

- dot (.) is a wildcard character – matches any character **except** new line (\n)

/a . c/ => 'a' followed by any character followed by 'c'

- Combine metacharacters

/a.{4}c/ => 'a' followed 4 instances of any character followed by 'c' so will match

- addddc
- Afgthc
- ab569c

Alternative Matching

- Match this **or** this.
- Two ways which depend on nature of pattern
- use a verticle bar '|' matches if either left side or right side matches,
- /(human|mouse|rat)/ => any string with human or mouse or rat.

Metacharacters

character class is a list of characters within '[]'. It will match any single character within the class. Brackets are used to find a range of characters.

/ [wxyz1234\t] / => any of the eight.

a range can be specified with '-'

/ [w-z1-4\t] / => as above

to match a hyphen it must be first in the class

/ [-a-zA-Z] / => any letter character or a hyphen

so only characters a to z and A to Z can be used

The stuff inside the square bracket can only be used

negating a character with '^' / [^z] / => any character except z

/ [^abc] / => any character except a or b or c

Character class

character class is a list of characters within '[]'. It will match any single character within the class. Brackets are used to find a range of characters:

/ [wxyz1234] / => any of the eight.

a range can be specified with '-'

/ [w-z1-4] / => as above

to match a hyphen it must be first in the class

/ [-a-zA-Z] / => any letter character or a hyphen

negating a character with '^' / [^z] / => any character except z

/ [^abc] / => any character except a or b or c

Metacharacters

Metacharacter	Description
<code>.</code>	Find a single character, except newline or line terminator
<code>\w</code>	Find a word character
<code>\W</code>	Find a non-word character
<code>\d</code>	Find a digit
<code>\D</code>	Find a non-digit character
<code>\s</code>	Find a whitespace character
<code>\S</code>	Find a non-whitespace character
<code>\b</code>	Find a match at the beginning/end of a word, beginning like this: <code>\bHI</code> , end like this: <code>HI\b</code>
<code>\B</code>	Find a match, but not at the beginning/end of a word
<code>\0</code>	Find a NULL character
<code>\n</code>	Find a new line character
<code>\f</code>	Find a form feed character
<code>\r</code>	Find a carriage return character
<code>\t</code>	Find a tab character
<code>\v</code>	Find a vertical tab character

Commonly used metacharacters

- `\d` => any digit `[0-9]`
- `\w` => any “word” character `[A-Za-z0-9_]`
- `\s` => any white space `[\t\n\r\f]`

- `\D` => any character except a digit `[^\d]`
- `\W` => any character except a “word” character `[^\w]`
- `\S` => any character except a white space `[^\s]`

- Can use any of these in conjunction with quantifiers,
- `/\s*/` => any amount of white space

Examples

- Only digits (min 4 max 6 chars)

[0-9]{4,6}

- Only lowercase, Uppercase, Numerals (min 4 max 10 chars)

[a-zA-Z0-9]{4,10}

- Only Decimal

[1-9]\d*(\.\d+)? ➔ 898.4

- Only whole numbers

[\d+]

Anchoring a Pattern

-
- Syntax: **/pattern/modifier**
 - Pattern – as explained in previous slides
 - Start indicator caret **“^”** (shift 6) marks the beginning of string
 - End indicator dollar **“\$”** marks end of the search pattern
 - Example : **^[a-z0-9_]{3,16}\$**
 - Modifier
 - Modifiers are used to perform case-insensitive and global searches:

Anchoring a Pattern

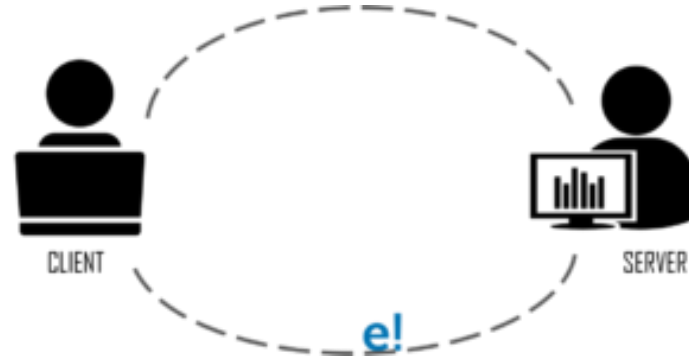
- Modifier

Modifier	Description
<code>g</code>	Perform a global match (find all matches rather than stopping after the first match)
<code>i</code>	Perform case-insensitive matching
<code>m</code>	Perform multiline matching

- Example string : “Nanyang Polytechnic SIT \nnanyang”
- `/nanyang/g` → no matches (g matches exact word)
- `/nanyang/i` → match “Nanyang” (case in-sensitive)
- `/nanyang/m` → match the first occurrence of Nanyang
- Modifiers can be combined

Input validation

- Form validation



- Client Side (Javascript, jQuery)

- Have to be dependent on browser and scripting language support.
 - Considered convenient for users as they get instant feedback.
 - The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.

- Server Side

- Uses ASP.NET control validators
 - For developer point of view server side is preferable because not dependent on browser and scripting language.

Javascript example

```
<!DOCTYPE html>
<html>
<body>

<p>Example - global modifier</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var str = "Nanyang Polytechnic SIT \nNanyang";
  var regx = /Nanyang/g;
  var result = str.match(regx);
  document.getElementById("demo").innerHTML = result;
}
</script>

</body>
</html>
```

Example - global modifier

Try it

Nanyang,Nanyang

Example

- Password complexity :

`/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[^\a-zA-Z0-9])(?!.*\s){8,15}$/`

- `(?=.*\d)` : at least a digit
- `(?=.*[A-Z])` : at least an uppercase letter
- `(?=.*[a-z])` : at least a lowercase letter
- `(?=.*[^\a-zA-Z0-9])` : at least a special char
- `(?!.*\s)` : no spaces
- `{8,15}` : between 8 t 15 chars

Email

`/^\\w+[\\+\\.\\w-]*@([\\w-]+\\.)*\\w+[\\w-]*\\.([a-z]{2,4}|\\d+)$/i`

/ = Begin an expression

^ = The matched string must begin here, and only begin here

\\w = any word (letters, digits, underscores)

+ = match previous expression at least once, unlimited number of times

[] = match any character inside the brackets, but only match one

\\+\\. = match a literal + or .

\\w = another word

- = match a literal -

* = match the previous expression zero or infinite times

@ = match a literal @ symbol

() = make everything inside the parentheses a group (and make them referencable)

[] = another character set

\\w- = match any word or a literal -

+ = another 1 to infinity quantifier

\\. = match another literal .

* = another 0 to infinity quantifier

\\w+ = match a word at least once

[\\w-]*\\. = match a word or a dash at least zero times, followed by a literal .

() = another group

[a-z]{2,4} = match lowercase letters at least 2 times but no more than 4 times

| = "or" (does not match pipe)

\\d+ = match at least 1 digit

\$ = the end of the string

/ = end an expression

i = test the string in a case insensitive manner

Valid Emails

- mkyong@yahoo.com
- mkyong-100@yahoo.com
mkyong.100@yahoo.com
- mkyong111@mkyong.com
- mkyong-100@mkyong.net
- mkyong.100@mkyong.com.au
- mkyong@1.com
- mkyong@gmail.com.com
- mkyong+100@gmail.com
- mkyong-100@yahoo-test.com
- <http://emailregex.com/>

InValid Emails

- mkyong – must contains “@” symbol
- mkyong@.com.my – tld can not start with dot “.”
- mkyong123@gmail.a – “.a” is not a valid tld, last tld must contains at least two characters
- mkyong123@.com – tld can not start with dot “.”
- mkyong123@.com.com – tld can not start with dot “.”
- .mkyong@mkyong.com – email’s first character can not start with dot “.”
- mkyong()*@gmail.com – email’s is only allow character, digit, underscore and dash
- mkyong@%*.com – email’s tld is only allow character and digit
- mkyong..2002@gmail.com – double dots “.” are not allow
- mkyong.@gmail.com – email’s last character can not end with dot “.”
- mkyong@mkyong@gmail.com – double “@” is not allow
- mkyong@gmail.com.1a -email’s tld which has two characters can not contains digit

End of Section 2

Next Session ...

Security Strategies
Security Principles

Secure Coding Techniques

Introduction

**Common Software
Vulnerabilities and
Controls**

Security Strategies

Security Principles

Security Strategies

- Security should be built into applications right from the start.
- Developers, designers, architects and project managers would do well to follow these tried and tested security principles.
- The principles are derived from the SD³+C strategies (from Microsoft).

Source: <http://msdn.microsoft.com/en-us/library/windows/desktop/cc307406.aspx>

SD³ + C Strategies

- Secure by Design
 - This means that developers follow secure coding best practices and implement security features in their applications to overcome vulnerabilities.
 - Ensure that the software design is secured right from the start.
 - Bad software design can make software difficult to secure later.
 - E.g: your application handles sensitive data, so you will encrypt the data and protect it from theft and tampering. This consideration to use cryptography in your application is done at the design stage.

SD³ + C Strategies

- Secure by Default
 - **Least privilege.** Allow each user/process minimum privileges to do his/its work.
 - **Defense in depth.** Design software that will not easily break down just because one security mechanism has been broken. If possible, employ multiple security mechanisms such that an attacker would need to breach several layers before being successful.
 - **Conservative default settings.** The development team is aware of the attack surface for the product and minimizes it in the default configuration
 - **Avoidance of risky default changes** to operating system or security settings
 - **Less commonly used services deactivated by default.**

SD³ + C Strategies

- Secure in Deployment
 - This means that applications can be maintained securely after deployment by updating with security patches, monitoring for attacks, and by auditing for malicious users and content.
 - Security functions must be easily administered by users.
 - Software must be easily patched to allow for security patches, if required.
 - In the event of application failure or errors, these events should be logged so that administrators can help resolve the issues.

SD³ + C Strategies

- Secure in Communications
 - **Security response:** Development teams responding promptly to reports of security vulnerabilities and communicate information about security updates.
 - **Community engagement:** Development teams proactively engage with users to answer questions about security vulnerabilities, security updates, or changes in the security landscape.

Secure Coding Techniques

Introduction

**Common Software
Vulnerabilities and
Controls**

Security Strategies

Security Principles

Security Principles

- Minimize your attack surface
 - Reduce potential points of entry from which attackers can take advantage of.
 - In particular, take note of the following:
 - Services running in elevated privilege
 - Services that are running by default
 - Applications that are running
 - Open ports
 - Open named pipes
 - Accounts with administrative rights
 - Files, directories and registry keys with weak access control lists (ACLs)

Security Principles

- Employ secure defaults
 - Most users would choose the defaults when installing your software.
 - Ensure default settings/services are necessary and not exploitable by hackers.
 - E.g. Windows 2000 installs IIS by default. This allows hackers to attack systems through the IIS.
- Assume external systems are insecure
 - For all data that is received from external sources, consider them to be possibly from attackers.
 - Filter these data for validity before allowing them into the system.

Security Principles

- Fail Safely
 - Design your program to recover or terminate 'gracefully' upon any form of failure.
 - When the application fails, ensure that data is not lost or disclosed to unauthorized parties.
- Remember that security features != security
 - Use correct security mechanisms to mitigate relevant threats.
 - Software will not be secure because a lot of security mechanisms are implemented.

Security Principles

- Never depend on security through obscurity alone
 - Always assume that the attacker knows everything you know.
 - Obscurity is a useful defense only when it is not the only defense.
 - Attackers can easily find information that you try to hide.

Secure Coding Techniques

The Problem

Secured Applications

- How to build Secure Software?
- Attacker's Advantage & Defender's Dilemma

Security Strategies

- Secure by Design
- Secure by Default
- Secure by Deployment

Security Principles

- Minimize your attack surface
- Employ secure defaults
- Assume external systems are insecure
- Fail Safely
- Security features != security
- Never depend on security through obscurity alone

End of Section 3

Next Session

Lesson 3.1