Secure Software Design and Engineering
(CY-321)

A Starter to Defensive Coding

Dr. Zubair Ahmad

Secure software is more than just writing secure code

Implementing controls in code can have a huge impact on the resiliency of the software against hacker threats

The moment a single line of code is written, What would happen?

Reducing the amount of code and services that are executed by default.

Reducing the volume of code that can be accessed by untrusted users.

limiting the damage when the code is exploited

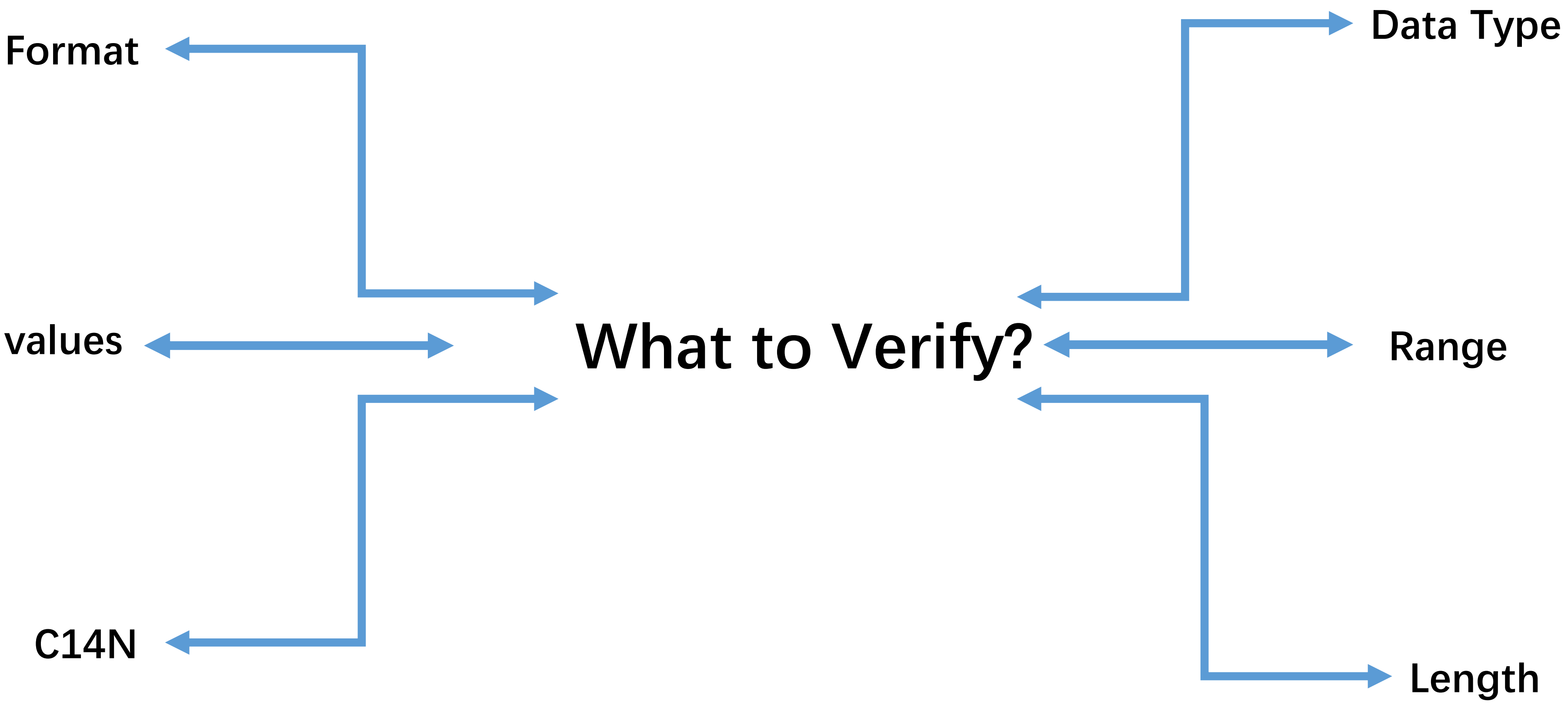
All inputs are evil

Consider all input as evil and validate all user input

Processing is of the correct data type and format

Falls within the expected and allowed range of values

Does not put in alternate forms that bypass security controls


Format
values
C14N
What to Verify?
Data Type
Range
Length

Where to Verify?

The point at which the input is validated is also critically important.

if the software is a Client/Server architected solution, both on the client (frontend) as well as on the server (backend)

Insufficient to validate input solely on the client side as this can be easily bypassed and afford minimal to no protection

Using RegEx

Powerful pattern-matching tools used to validate input data by enforcing rules for format, structure, and allowed characters

Using RegEx

Email Addresses

Pattern: Ensures correct structure like user@example.com

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

Using RegEx

Email Addresses

```
import re

email_pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
email = "user@example.com"

if re.match(email_pattern, email):
    print("Valid email")
else:
    print("Invalid email")
```

Using RegEx

Passwords

Pattern: At least **8 characters**, **1 uppercase, 1 lowercase, 1 number, and 1 special character**.

```
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$
```

Using RegEx

Passwords

```
password_pattern = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
password = "Strong@123"

if re.match(password_pattern, password):
    print("Valid password")
else:
    print("Invalid password")
```

Canonicalization (C14N)

We call a name “canonical” if two names that denote the same object have the same canonical name.

Canonicalization

```python
def canonicalize_string(text):
    return " ".join(text.strip().lower().split())

print(canonicalize_string("  Hello   WORLD  "))
print(canonicalize_string("HELLO WORLD"))
```

Canonicalization

```
from urllib.parse import urlparse, urlunparse

def canonicalize_url(url):
    parsed = urlparse(url)
    scheme = parsed.scheme.lower()
    netloc = parsed.netloc.lower()
    path = parsed.path.rstrip('/')
    return urlunparse((scheme, netloc, path, 
"", "", ""))

print(canonicalize_url("HTTPS://Example.COM/Ho
me/"))
```

Canonicalization

```
import os

def canonicalize_path(path):
    return
os.path.abspath(os.path.normpath(path)
)

print(canonicalize_path("../myfolder
/../file.txt"))
```

Canonicalization

```
import json

def canonicalize_json(data):
    return json.dumps(data, sort_keys=True,
separators=(',', ':'))

data = {"b": 2, "a": 1}
print(canonicalize_json(data))
)
```

Sanitizing the Input

Converting something that is considered dangerous into its innocuous form

Do we need to Sanitize the Output as well?

Sanitizing the Input

Stripping: Removing harmful characters from user supplied input

Substitution: Replacing user supplied input with safer alternatives

Literalization: Using properties that render the user supplied input to be treated as a literal form

Sanitizing the Input

Stripping

Hello! <script>alert('Hacked!');</script>

```
import re

def strip_html_tags(input_text):
    return re.sub(r'<[^>]*>', '', input_text)

user_input = "Hello! 
<script>alert('Hacked!');</script>"
sanitized_input = strip_html_tags(user_input)
print(sanitized_input)
```

Sanitizing the Input

Substitution

```
import pymysql

def safe_query(user_input):
    safe_input = user_input.replace("'", "''")  # Escape single
quotes
    query = f"SELECT * FROM users WHERE username = '{safe_input}'"
    return query

user_input = "admin' --"
sanitized_query = safe_query(user_input)
print(sanitized_query)
# Output: SELECT * FROM users WHERE username = 'admin'' --'
```

Sanitizing the Input

Lateralization

```
import sqlite3

conn = sqlite3.connect(":memory:")
cursor = conn.cursor()
cursor.execute("CREATE TABLE users (id INTEGER PRIMARY
KEY, username TEXT)")

def get_user(username):
    query = "SELECT * FROM users WHERE username = ?"
    cursor.execute(query, (username,))  # Parameterized
query
    return cursor.fetchall()

user_input = "admin' OR 1=1 --"
result = get_user(user_input)
print(result)
```

Questions??

zubair.ahmad@giki.edu.pk

Office: G14 FCSE lobby