Common Software Vulnerabilities

Software Vulnerabilities [1 of 2]

Oregon State University
College of Engineering

- Software vulnerabilities are flaws or weaknesses present in software
  - Application or OS software
  - Potentially exploitable

Software Vulnerabilities [2 of 2]

Oregon State University
College of Engineering

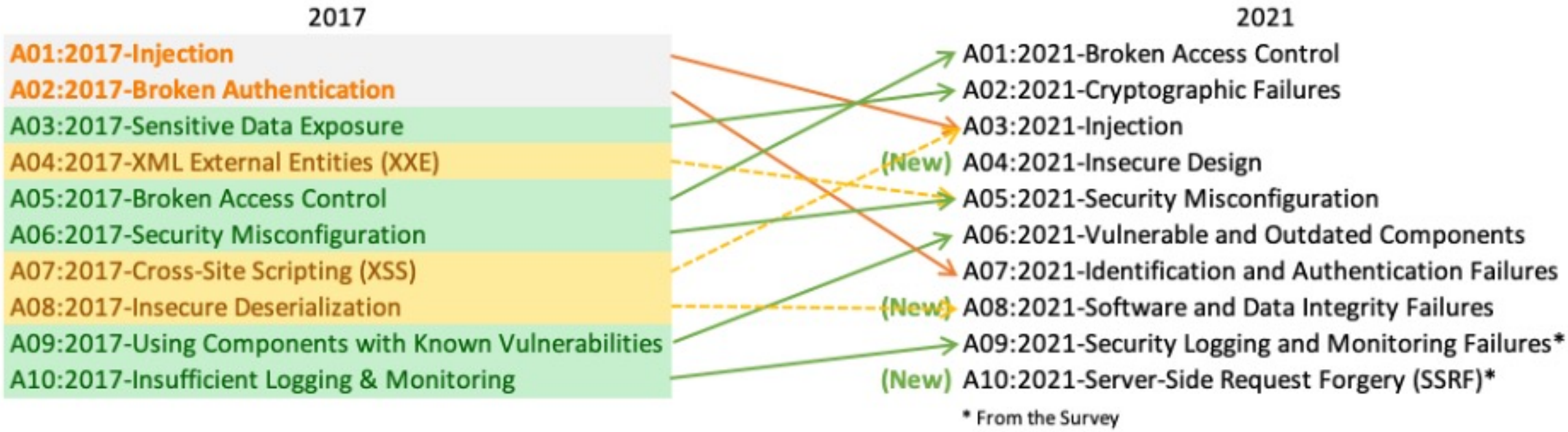
- Generally, a result of poor programming practices

- List of common vulnerabilities are maintained
  - OWASP top 10 Web Application Security risks was updated in 2021
  - OWASP top 10 Mobile
    - Find them at OWASP.org
  - CWE/SANS top 25 is from 2011
    - Find it at cwe.mitre.org
  - A good number are developer bugs

- Good coding practice lists/guides are also available (e.g., see OWASP Top 10 Proactive Controls)

OWASP Top 10

Oregon State University
College of Engineering



Software Vulnerabilities

Oregon State University
College of Engineering

- We'll classify them in broader categories
  - So we can identify better these problems and future ones

Broader Categories of Software Vulnerabilities

Oregon State University
College of Engineering

- Program Input (Input Checking)
  - e.g., code and data injection
- Program Code (Program Logic Errors)
  - e.g., broken access control, bad random numbers or seeds
- Interacting with Operating systems and other Programs
  - e.g., memory leaks, race conditions, environment variables
- Program Output (Output Checking)
  - e.g., cross site scripting (XSS)

Security in Design and Architecture

Oregon State University
College of Engineering

- Security concerns must be considered up front

- Security impacts system architecture
  - Perhaps re-architect to ameliorate security concerns
  - Security architecture can be used to drive testing
  - True even for projects with no “security features”

- Leaving security to the test phase (or later) is not good
  - End up patching a few holes without knowing whether all the problems are found
  - Often, one ends up with much more work to do

Defensive Programming or Secure Coding

Oregon State University
College of Engineering

- How do we write secure programs?

- Mostly good software engineering
  - However, security != reliability
  - When considering security must consider a malicious actor
  - Traditional software engineering concentrates dealing with errors due to accidents

- Goal
  - Continued functioning of software in spite of unforeseeable

- Conflicts with time to market

Make No Assumption

Oregon State University
College of Engineering

- Successful attacks exploit **implicit assumptions** made by programmers
  - "The user needs to enter a phone number. It surely will be at most 15 digits."
  - "A user sent a piece of text and I need to visualize in somebody else's browser. It surely will be just text, no need of doing anything with it."
  - Don't assume user won't enter > 512 characters on the command line
  - Don't assume that there will always be enough disk space

- **Make no (implicit) assumptions!!**
  - Everything you assume needs to be codified and enforced

Summary

Oregon State University
College of Engineering

- Software vulnerabilities are a key factor in security breaches
- Many software vulnerabilities are a result of poor programming practices
- List of good controls or secure programming practices is available
- Secure programming is mostly good software engineering but needs to account for adversaries instead of just accidental errors
- All assumptions when designing code/systems should be made explicit

Software Vulnerabilities - I

Software Vulnerability Categories

Oregon State University
College of Engineering

- Program Input (Input Checking)
  - e.g., code and data injection
- Program Code (Program Logic Errors)
  - e.g., broken access control, bad random numbers or seeds
- Interacting with Operating systems and other Programs
  - e.g., memory leaks, race conditions, environment variables
- Program Output (Output Checking)
  - e.g., cross site scripting (XSS)

Oregon State University
College of Engineering

Input Handling Vulnerabilities

Handling Input

Oregon State University
College of Engineering

- This is where the user (malicious/innocent) directly impacts program
  - Always verify the user input
  - Whitelist expected results
- Input can come from a number of places
  - Text entry
  - Configuration files
  - Environment variables
  - Network

Injection Attacks

Oregon State University
College of Engineering

- Error in input handling that results in unexpected execution flow (Another view: Untrusted input that results in unexpected execution flow)
  - Script writer expects user input to be data
  - But user inputs text that will be interpreted as code
  - Scripting languages are particularly vulnerable
- A few examples of injection attacks:
  - Code injections (i.e., buffer overflows)
  - Command injections
  - SQL injections

Unsafe Wrapper Code

Oregon State University
College of Engineering

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {

        char cat[] = "cat ";
        char *command;
        size_t commandLength;

        commandLength = strlen(cat) + strlen(argv[1]) + 1;
        command = (char *) malloc(commandLength);
        strncpy(command, cat, commandLength);
        strncat(command, argv[1], (commandLength - strlen(cat)) );

        system(command);

        return (0);
}
```

Example from OWASP - https://www.owasp.org/index.php/Command_Injection

Running Wrapper Code

Oregon State University
College of Engineering

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt       doubFree.c   nullpointer.c
unstosig.c      www*         a.out*
format.c        strlen.c     useFree*
catWrapper*     misnull.c    strlength.c   useFree.c
commandinjection.c           nodefault.c   trunc.c       writeWhatWhere.c
```

- **Command injection**. Running arbitrary commands at the privilege of the web user id.

Example from OWASP - https://www.owasp.org/index.php/Command_Injection

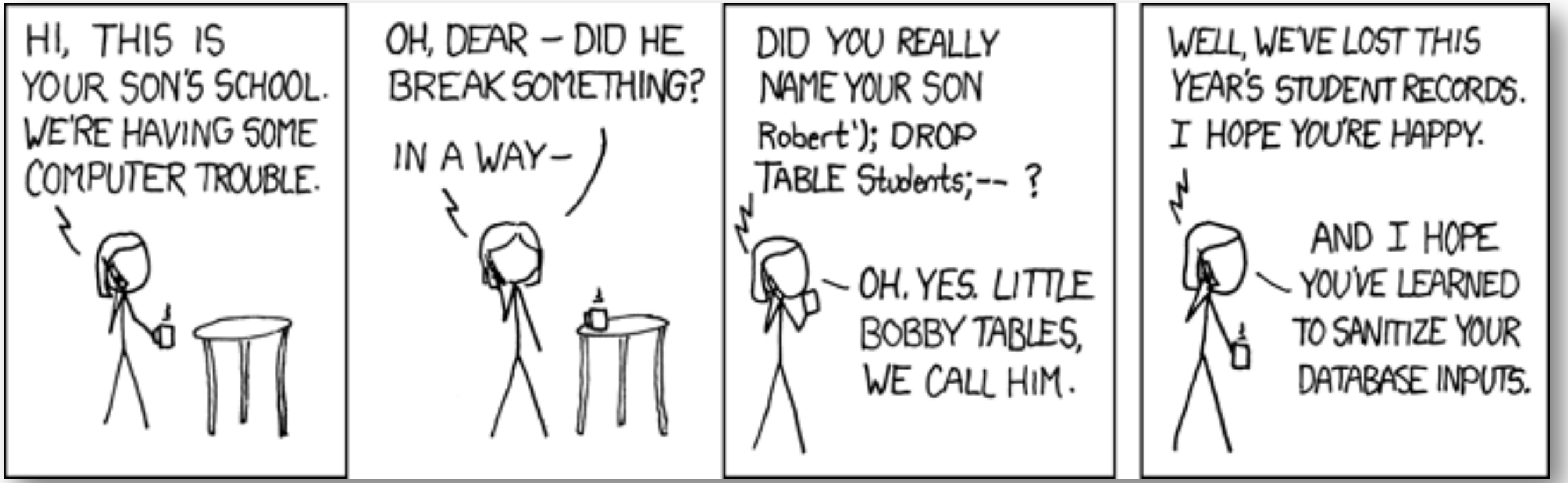
Safer Code

Oregon State University
College of Engineering

- Counter the attack by validating input
  - compare to pattern that rejects invalid input
  - **Do not** search for bad inputs, ensure pattern **only accepts** valid input

SQL Injection

Oregon State University
College of Engineering



SQL Injection

Oregon State University
College of Engineering

- Another widely exploited injection attack
- When input is used in SQL query to database
  - similar to command injection
  - SQL meta-characters are the concern
  - must check and validate input for these

SQL Injection Example

Oregon State University
College of Engineering

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
               + userName + "' AND itemname = '"
               + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

SQL Query

```
SELECT * FROM items
WHERE owner =
AND itemname = ;
```

Source: https://owasp.org/www-community/attacks/SQL_Injection

SQL Injection Example

Oregon State University
College of Engineering

If a user inputs itemname to be “name’ OR ‘a’ = ‘a”

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

SQL Query

```
SELECT * FROM items;
```

Source: https://owasp.org/www-community/attacks/SQL_Injection

Code Injection

Oregon State University
College of Engineering

- Further variant
- Input includes code that is then executed
  - this type of attack is widely exploited

```
$myvar = "varname";
$x = $_GET['arg’];
eval("\$myvar = \$x;");
```

```
/index.php?arg=1; phpinfo()
```

Cross Site Scripting (XSS)

Oregon State University
College of Engineering

- Goal – Inject malicious code into web pages viewed by others.
  - Sites that allow HTML formatted user input to be stored
  - e.g. Blog comments, wiki entries.

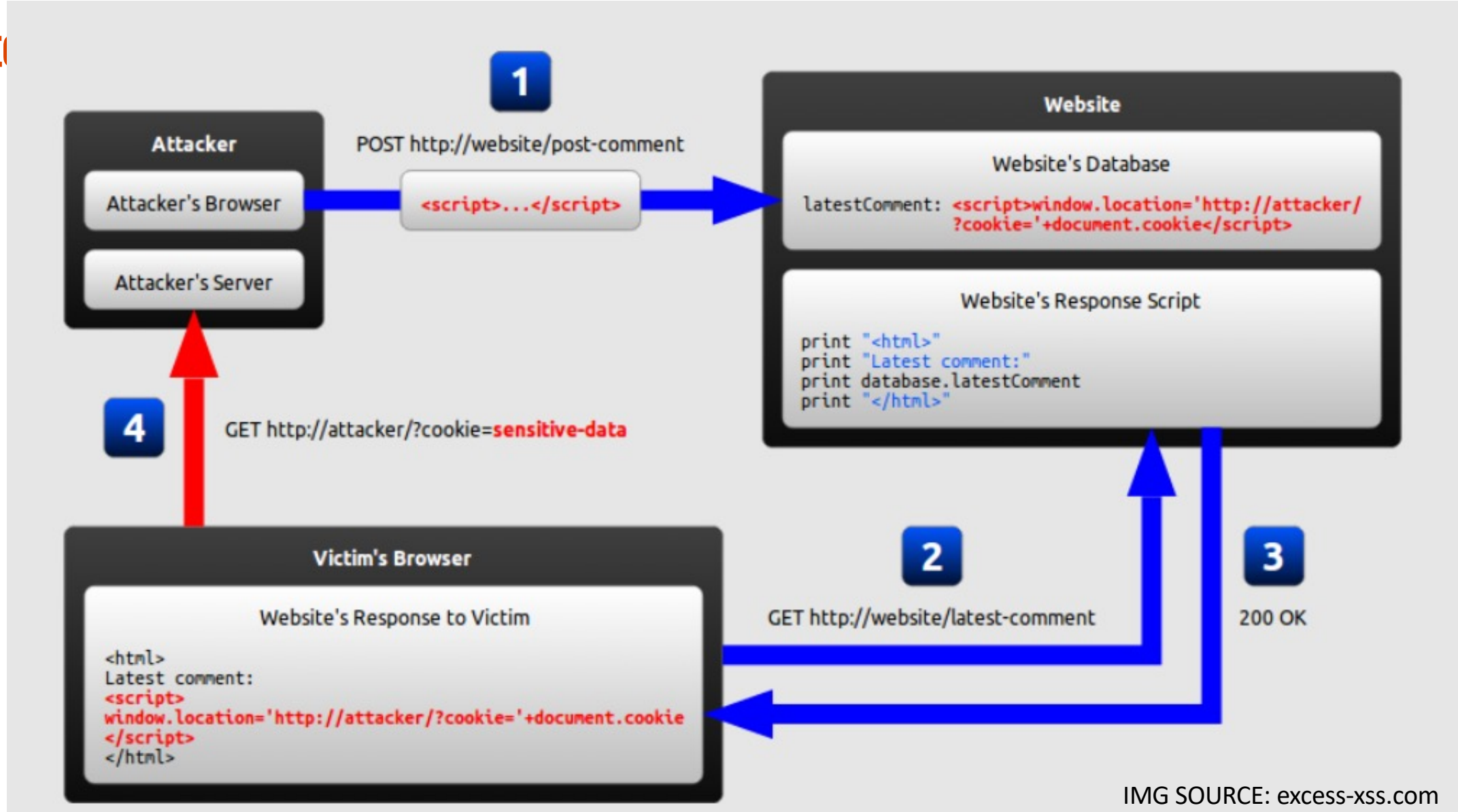
XSS Example

Oregon State University
College of Engineering

- cf. guestbooks, wikis, blogs etc where comment includes script code
  - e.g. to collect cookie details of viewing users
- need to validate data supplied
  - including handling various possible encodings
- attacks both input and output handling

- "script" is persistent in the website

Persist



IMG SOURCE: excess-xss.com

Input Checks

Oregon State University
College of Engineering

- Canonicalize input before performing checks
  - Map the multiple versions of 'A' to a particular value
- Issue for numeric values too
  - Is the number 16 bits or 32?
  - Signed or unsigned?
    - Negative number or large positive
- Use good libraries:
  - PreparedStatement in java, OWASP ESAPI toolkit, and many others

Input Fuzzing

Oregon State University
College of Engineering

- Can we test a program to find bad input processing?

- Generate "random" inputs to test programs
  - Environment variables
  - Input strings
  - Network values

- Could be completely randomized or somewhat structured
  - Minifuzz
  - ShareFuzz
  - Spike
  - MuDynamics

- Standard component of Microsoft's Software Development Lifecycle

Summary

Oregon State University
College of Engineering

- Correct input handling is essential for programs to be secure
- Improper input handling could lead to code, data and command injection attacks
  - Example: SQL Injection, Cross Site Scripting, Code Injection, Buffer overflows
- Input fuzz testing can help identify parts of a programs that are vulnerable to input handling vulnerabilities