Input Validation Introduction

Target Course

CS1

Learning Goals

A student shall be able to:

1. Describe security design principles and identify security issues associated with common threats and attacks.

IAS Outcomes

The CS2013 Information Assurance and Security outcomes addressed by this module are:

IAS Knowledge Topic	Outcome
Defensive Programming	1. Explain why input validation and data sanitization is necessary in the face of adversarial control of the input channel. [Familiarity] 3. Classify common input validation errors, and write correct input validation code. [Usage]
Principles of Secure Design	2. Summarize the principle of fail-safe and deny-by-default. [Familiarity]

Dependencies

- Assumes no pre-requisite knowledge.

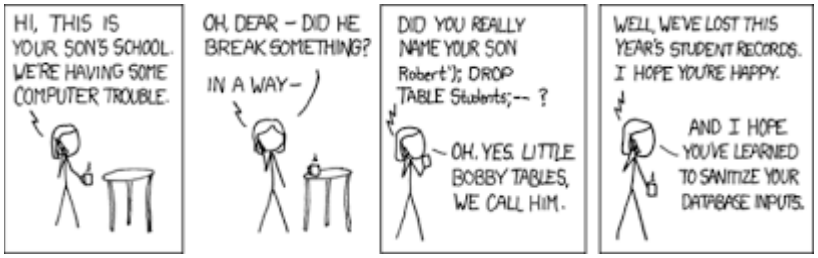
Summary

Introduce the need for input validation and data sanitization to help prevent both incorrect results from being produced and adversarial use of the input channel that results in unauthorized access to data.

Estimated Time

[Provide the estimated amount of lecture time to cover this module, using the notion of time as defined in CS2013.]

Materials



Why is input validation important?

Programs often use external data which is entered by a user, or read in from a file, database, or a network connection. When trusted users accidently enter incorrect or unexpected input data it can cause programs to crash or produce results that are not what was expected (see comic strip above). Some real-life examples of such occurrences are:

- In December 2005, a Japanese securities trader made a $1 billion typing error, when he mistakenly sold 600,000 shares of stock at 1 yen each instead of selling one share for 600,000 yen. A few lines of code may have averted this error. Fat fingered typing cost a trader's boss £128m. [1]

- A Norwegian woman mistyped her account number on an internet banking system. Instead of typing her 11-digit account number, she accidentally typed an extra digit, for a total of 12 numbers. The system discarded the extra digit, and transferred $100,000 to the (incorrect) account. A simple dialog box informing her that she had typed too many digits would have helped avoid this expensive error. [2]

An *untrustworthy* or *malicious* user is one who will intentionally craft input data to cause programs to run unauthorized commands, leading to security vulnerabilities. Web applications are especially prone to SQL injection. Here a malicious user crafts an input containing some SQL command which if executed causes the application to reveal private or unauthorized information (e.g. to dump the database contents) as was the case in the following real-life example:

- In Feb 2002, Jeremiah Jacks discovered that Guess.com was vulnerable to an SQL injection attack. A properly-crafted URL allowed anyone to pull down 200,000+ names, credit card numbers and expiration dates in the site's customer database. [3]

Other common attacks that leverage poor input validation include:

- Cross-site scripting – allows attackers to inject client side scripts into webpages to bypass access controls.
- In-band signaling attacks- Older phone networks used in-band signaling to send phone commands and voice data over the same channel. It allowed for phone phreaking attacks where phone freaks intentionally supplied commands intended for testing and administrative use to make free long-distance calls.

How should input data be handled?

Since input data can lead to security risks, program failures, and/or incorrect results, input data should be validated before it is used by the program. This agrees with the **fail safe** design principal, which states that systems should be designed so that in the case of errors the safest outcome should be produced. The following is a list from Towson University's security injection modules of some of the most common ways to validate input data [4]:

- **Type check:** input should be checked to ensure it is the data type expected, e.g., age must be integer.
- **Range check (reasonableness check)** - numbers checked to ensure they are within a range of possible values, e.g., the value for month should lie between 1 and 12.
- **Length check:** variables are checked to ensure they are the appropriate length, for example, a US telephone number has 10 digits.
- **Format check** – Check that the data is in a specified format (template), e.g., dates might be required to be in the format DD/MM/YYYY.
- **Arithmetic Errors:** variables are checked for values that might cause problems such as division by zero or integer overflow.

What should be done if erroneous input is detected?

If an input contains errors the program should immediately reject that request. It should not attempt to interpret erroneous input into something meaningful. While the approach of rejecting inputs with errors may decrease usability (from a user's perspective, small errors require reentering the input), it increases security.

This approach agrees with the **deny-by-default** design principal, which states that everything not explicitly permitted is forbidden.

Assessment Methods

The answer to each question is in **bold**.

Multiple Choice

1. What does it mean to do type checking on an input data value?
   a. The input data value is validated to ensure the user can type correctly.
   b. The input data value is considered valid only when its value is of a certain type.
   c. The input data value is considered invalid only when its value is of a certain type.
   d. All of the above.

2. What does it mean to do range checking on an input data value?
   a. The input data value is validated to ensure the user can sing in multiple octaves.
   b. The input data value is considered invalid only when its value is within a certain range.
   c. The input data value is considered valid only when its value is within a certain range.
   d. All of the above.

3. Why is it a good idea to always validate data input by a user?
   a. The user may attempt to enter data with the intent of gaining access to systems/data that the user does not have the authority to use/see.
   b. The user may enter data that causes the program to crash.
   c. The user may enter data that causes the program to produce unpredictable results.
   d. All of the above.

4. Which of the following describes range checking?
   a. Ensuring that a date value looks like yyyy-mm-dd.
   b. Ensuring that a value is numeric.
   c. Ensuring that a numeric value is positive.
   d. Ensuring that a US phone value has ten digits.
   e. None of the above.

5. Which of the following describes length checking?
   a. Ensuring that a value is numeric.
   b. Ensuring that a US phone value has ten digits.
   c. Ensuring that a date value looks like yyyy-mm-dd.
   d. Ensuring that a numeric value is positive.
   e. None of the above.

6. Which of the following describes type checking?
   a. Ensuring that a US phone value has ten digits.
   b. Ensuring that a numeric value is positive.
   c. Ensuring that a date value looks like yyyy-mm-dd.
   d. Ensuring that a value is numeric.
   e. None of the above.

7. Which of the following describes format checking?
   a. Ensuring that a US phone value has ten digits.
   b. Ensuring that a value is numeric.
   c. Ensuring that a date value looks like yyyy-mm-dd.
   d. Ensuring that a numeric value is positive.
   e. None of the above.

8. Which code snippet is an example of arithmetic check?

a. Ensuring that a data value is numeric.
b. Ensuring that a numeric value is positive.
c. Ensuring that a US phone value has ten digits.
d. Ensuring that a date value looks like yyy-mm-dd.
e. None of the above.

True/False

9. True or **false**? When bad input data has been detected, the program code should try to correct the error to avoid having the user enter the value again.

References

[1] The Times Online. Slip of the fat finger. Retrieved April 2017 from http://www.thetimes.co.uk/article/slip-of-the-fat-finger-jk0zl9bhwgm.

[2] K. Olsen, "The $100,000 Keying Error," *IEEE Computer,* August 2008.

[3] "Guesswork Plagues Web Hole Reporting," *Security Focus,* 6 March 2006.

[4] Security Injection @ Towson. Retrieved August 2015 from *http://cis1.towson.edu/~cssecinj/modules/course-metrics/.*

[5] “Top 10 Secure Coding Practices” SEI CERT Coding Standards. Retrieved August 2015 from https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices.