Secure Information Flow

CIS-3720

Peter Chapin

Input Validation

- Input Validation: Verify the format *and constraints* on all inputs
  - Data entered manually by the user
  - Data taken from the command line or environment
  - Data read from files
  - Data read from the network
  - GUI events (mouse clicks, window events, etc.)
- Tool: Regular Expressions
  - Match input to check format.
  - Probably still requires constraint checks (although complex REs may be able to capture some constraints).

Input Validation

- Input Validation is about *data integrity*
  - Malicious user can't easily "drive" the program using bad inputs to force bad outputs.
  - Input validation *protects the integrity of the data **written** by the program*.
  - Input validation also *protects against program crashes*
    - Avoids denial of service attacks
  - Input validation *improves reliability*
    - Major tool for reliability enhancement

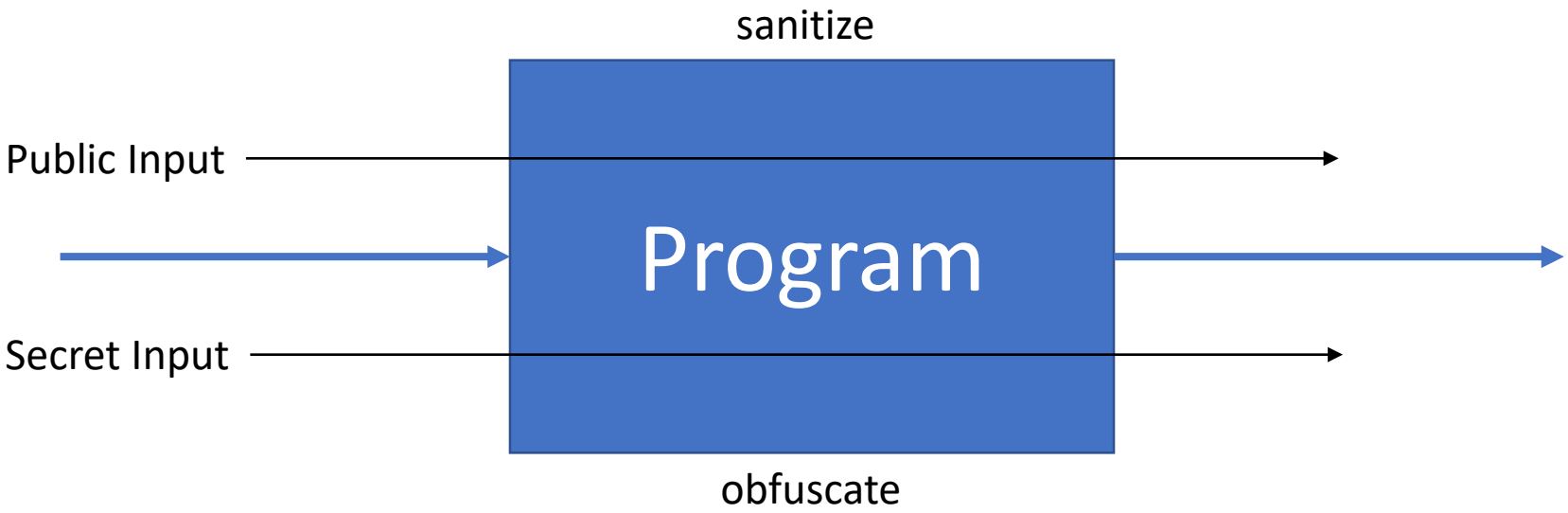
Confidentiality?

- Confidentiality is the *dual* of data integrity
  - Must not be able to force the program to let the user **read** secrets (confidentiality protection)
    - Program does not “leak” secret information
  - Must not be able to force the program to **write** outputs inappropriately (data integrity protection)
    - Program never outputs garbage
- Input validation only covers data integrity!
  - To protect data integrity: We must sanitize public input
  - To protect confidentiality: We must obfuscate private input

Example: Gradebook

- It is a violation of FERPA rules to let students see each other's grades
  - Suppose a gradebook program shows:
    - His/Her grade
    - Class average
  - Suppose there are only two students in the class
    - Jill sees: Grade = 84, Class average = 87
    - Jill calculates other grade: (84 + X)/2 = 87; **X = 90**
    - *Security violation!*
  - *Program did not properly obscure other grades; leaked secret information*

General Form



Let's Talk About Confidentiality

- Imagine four "security levels"
  - Unclassified (0), Sensitive (1), Secret (2), Top Secret (3)
  - Of course, we could use just two levels if we wanted
- Simple combination rules:
  - When level *x* "meets" level *y*, the result level is *max(x, y)*
  - That is: *the secrecy of the combined information is that of the highest component*
  - A security level can only be lowered by going through an "obfuscation function" defined by the developer.

Example…

- Consider the following code

```
int x = getFromUnclassfiedFile(); // Level 0
int y = getFromSecretFile();      // Level 2
int z = getFromTopSecretFile();   // Level 3
…
a = x + 1;  // Level 0 (constants don't affect level)
b = a + y;  // Level 0 and Level 2 results in Level 2
c = (2*a) / (b + z);  // Level ?
print(c);   // WARNING! Printing top secret information.
```

Example (continued)…

- Consider the following code

```
int x = getFromUnclassfiedFile(); // Level 0
int y = getFromSecretFile();      // Level 2
int z = getFromTopSecretFile();   // Level 3
…
a = x + 1;  // Level 0 (constants don't affect level)
b = a + y;  // Level 0 and Level 2 results in Level 2
c = (2*a) / (b + z);  // Level ?
c = obscuringMethod(c);  // Reduces to Level 0
print(c);   // Printing unclassified information.
```

Types?

- Notice that obscuringMethod takes a top secret parameter and returns an unclassified result. How do we declare it?
  - `level0 int obscuringMethod( level3 int param );` ?
  - Here we assume the language is extended with type qualifiers such as level0, level1, level2, etc.
- New type checking rules:
  - Every variable has a level (perhaps with a default)
  - Level of result is the maximum of input levels

Dynamic Security Levels?

- In Java and many languages, types don't change
  - Once declared as an int, always an int
- Should security levels work the same way?
  - Consider: c = obscuringMethod(c);
    - If obscuringMethod returns Level 0, does this entail storing a Level 0 value into a Level 3 variable? If so, it won't help the later print.
    - … or does the level of c change here?
    - … or do we have to use a different, level 0 variable to receive the result?
- Note: Many languages (Python) have dynamic types
  - … so dynamic levels wouldn't be weird in such a language.

Control Dependencies

- Consider the following code
  - ```
    int x = getFromUnclassfiedFile(); // Level 0
    int y = getFromSecretFile();      // Level 2
    int z = getFromTopSecretFile();   // Level 3
    …
    a = 0;
    if (z < 0) {
      a = x + 1;  // Level 0 (constants don't affect level)
    }
    print(a);   // WARNING! Printing top secret information.
    ```
- What level can you declare for a?
  - Security type systems tend to cause migration toward higher levels
    ```

Consider Arrays…

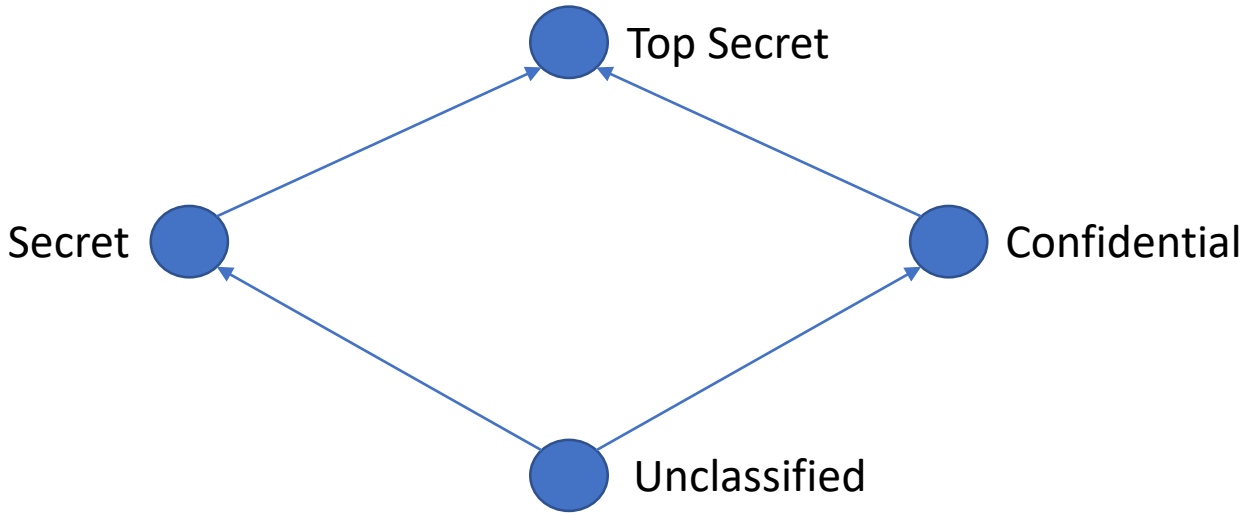
- Consider the following code
  - ```
    int x = getFromUnclassfiedFile(); // Level 0
    int y = getFromSecretFile();      // Level 2
    int z = getFromTopSecretFile();   // Level 3
    int[] array = new int[z];
    …
    print(array.length());  // WARNING! Top secret!
    Array[0] = x;
    print(array[i + 2*j – k]); // What level?
    ```
    ```

Consider Classes…

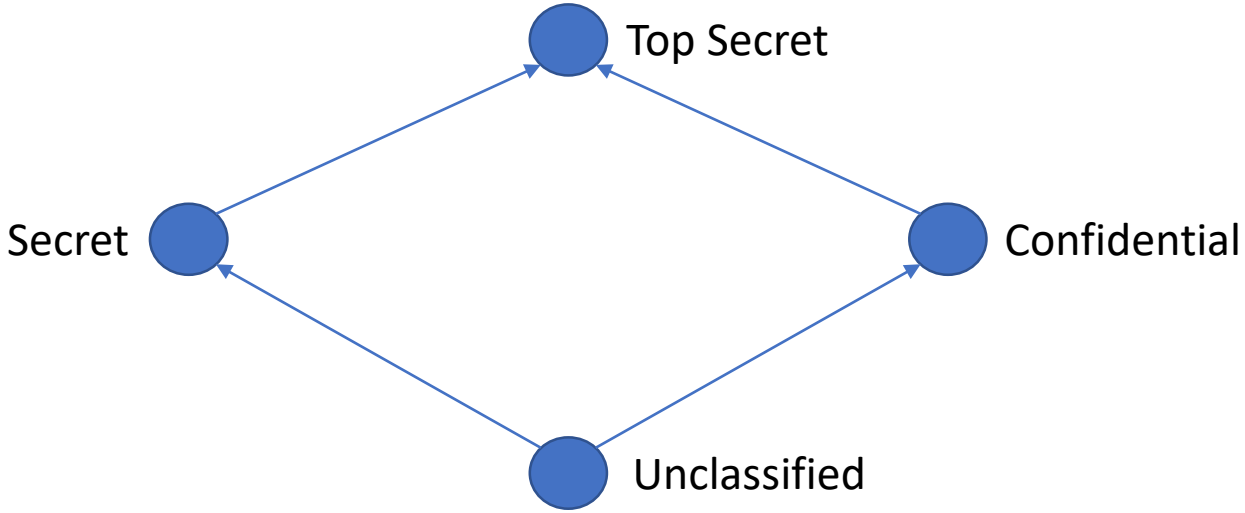
- Consider the following code
  - ```
    int x = getFromUnclassfiedFile(); // Level 0
    int y = getFromSecretFile();      // Level 2
    int z = getFromTopSecretFile();   // Level 3
    SomeClass s = new SomeClass(x, y, z);
      // Class contains top secret information.
    …
    print(s.getSomeValue());  // Top secret?
    ```
    ```

Incomparable Levels

- Suppose you had Unclassified, Secret, Confidential, Top Secret?
  - How do Secret and Confidential combine? Maybe they don't…



Least Upper Bound

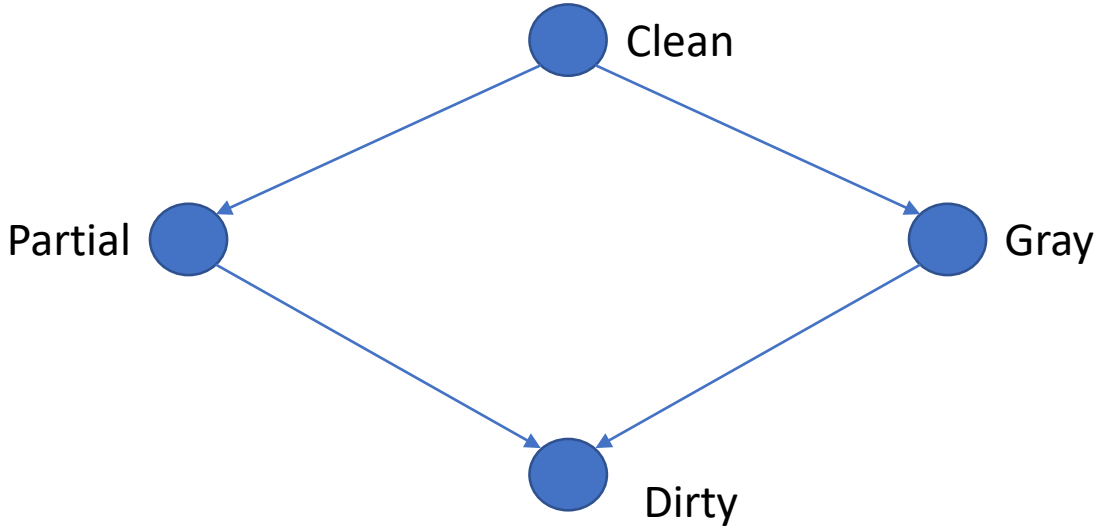


- Security, S, of result is the least upper bound…
  - S(Secret, Confidential) = Top Secret
  - S(Unclassified, Secret) = Secret
  - S(Unclassified, Top Secret) = Top Secret

In General…

- Security levels form a mathematical object called a “lattice”
  - Combined security is the LUB (least upper bound) of component levels
  - Security levels flow upwards
  - Obfuscation functions lower security level
  - Output must be at some predefined low level (unclassified?)
  - Different outputs have different requirements
- Program must trace security levels over the control flow
  - Either statically using a type system of some kind…
  - … or dynamically at run time
- ***This is hard!!***

Now the Dual… Taintedness



- Taintedness, T, of result is greatest lower bound
  - T(Partial, Gray) = Dirty
  - T(Partial, Clean) = Partial
  - T(Clean, Dirty) = Dirty

In General…

- Taintedness levels form a mathematical object called a “lattice”
  - Combined taintedness is the GLB (greated lower bound) of component levels
  - Taintedness levels flow downwards
  - Sanitization functions cleanse data
  - Output must be at some predefined high level (Clean?)
  - Different outputs have different requirements
- Program must trace taintedness levels over the control flow
  - Either statically using a type system of some kind…
  - … or dynamically at run time
- ***This is hard!!***

Perl

- Perl's *taint mode* is a dynamic taintedness check with only two levels
  - Uses REs to match format
  - Does not deal with high level constraints
  - Does not deal with confidentiality issues
  - Has runtime cost
- Simple, but limited

Traditional Input Validation

- Input validation attempts to de-taint (sanitize) input *immediately*
  - … and then assumes all other data in the program is clean.
  - Often workable
    - In contrast, obscuring confidential input immediate is often impractical
  - … but not always
  - Does nothing about confidentiality (secret leaking)

The “Right” Way

- Secure Information Flow is a research topic
  - Type systems tend to not work
    - You usually have to declare too much at a high security level
    - They also require language extensions
  - Static checking is hard
    - In general, undecidable
    - Requires specialized tools
  - *There is no ideal solution currently!*

Impractical Theory

- There is a concept of “information separation”
  - Show that the public output is not affected by any secret input.
  - Show that the critical (secret) output is not affected by any public input.
  - *Total separation!*
- Impractical…
  - Real programs routinely want to use secret information to impact public outputs. Often that is the very point of the program!
  - Consider: gradebook example showing class averages.