Please check your attendance using Blackboard!

Lecture 6 – Secure Software Development Lifecycle

[COSE451] Software Security

Instructor: Seunghoon Woo

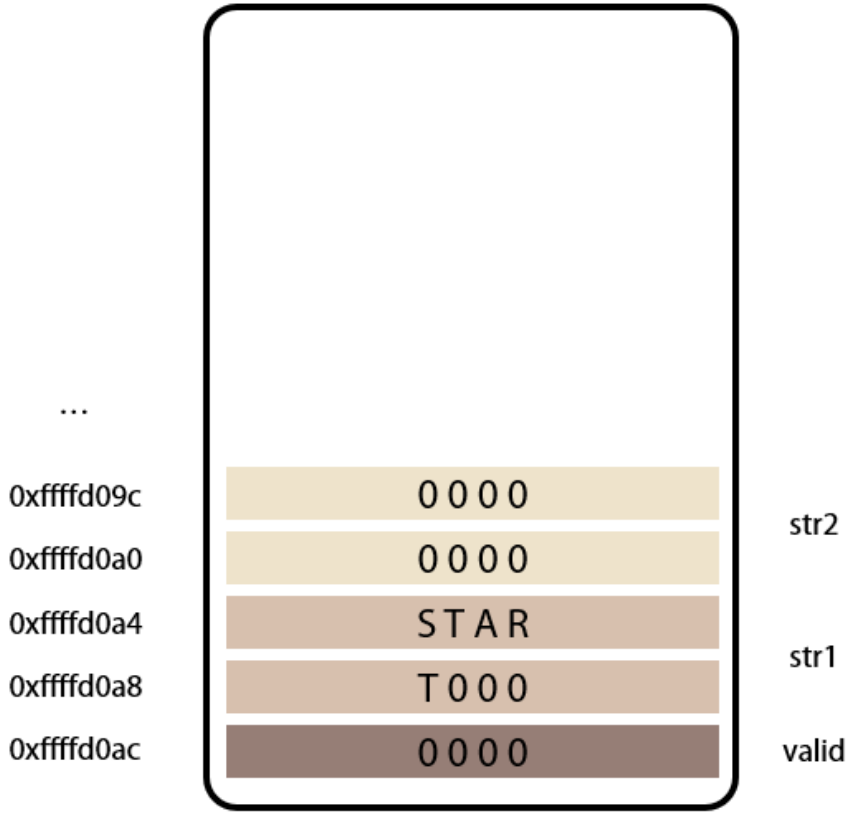
Spring 2024

Revisit..

Memory safety

- **Stack buffer overflow**
  - Example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char * argv[]) {
5      int valid = 0;
6      char str1[8] = "START";
7      char str2[8];
8
9      gets(str2);
10     if (strncmp(str1, str2, 8) == 0)
11         valid = 1;
12
13     printf("Buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
14
15 }
```

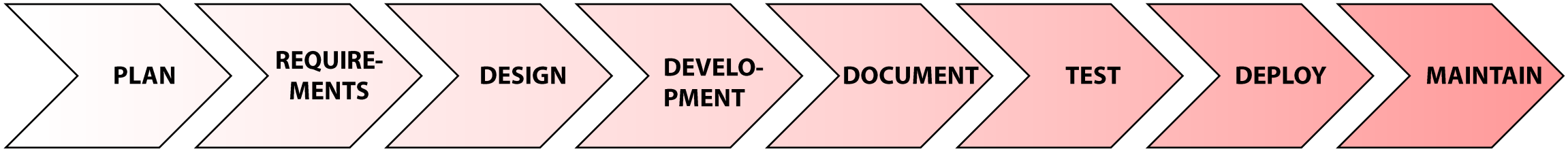


Overview

- **Secure Software Development Lifecycle**

Software Development Lifecycle

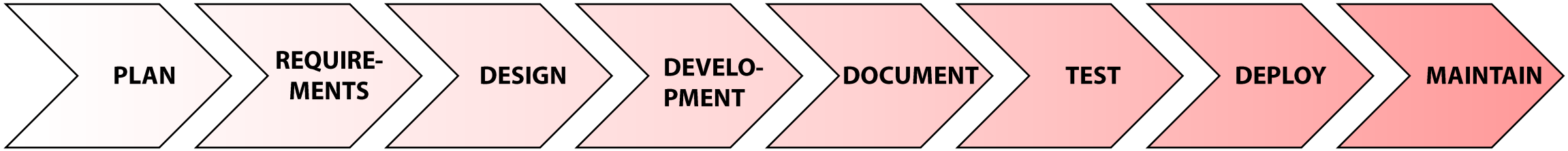
- **The process of planning, creating, testing, and deploying information systems across hardware and software**
  - Formalizing the tasks into 6-8 phases with the goal to improve software quality
    - The phases may vary depending on the company and purpose
    - Waterfall, V-shaped model, Agile, Sprial, etc.



https://www.redhat.com/en/topics/security/software-development-lifecycle-security

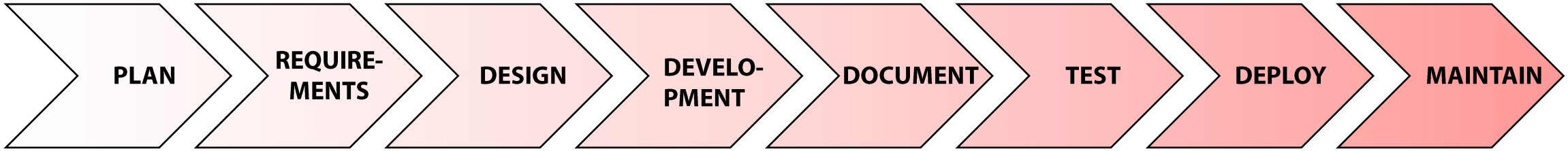
Software Development Lifecycle

Plan	Determine the scope and purpose of the software
Requirements	Define what functions the software should perform
Design	Decide key parameters like architecture, platforms, and user interfaces
Development	Create and implement the software
Document	Produce the information to help stakeholders understand how to use and operate the software
Test	Validate that the software fulfills the requirements
Deploy	Make the software available to its intended users
Maintain	Resolve bugs or vulnerabilities discovered in the software



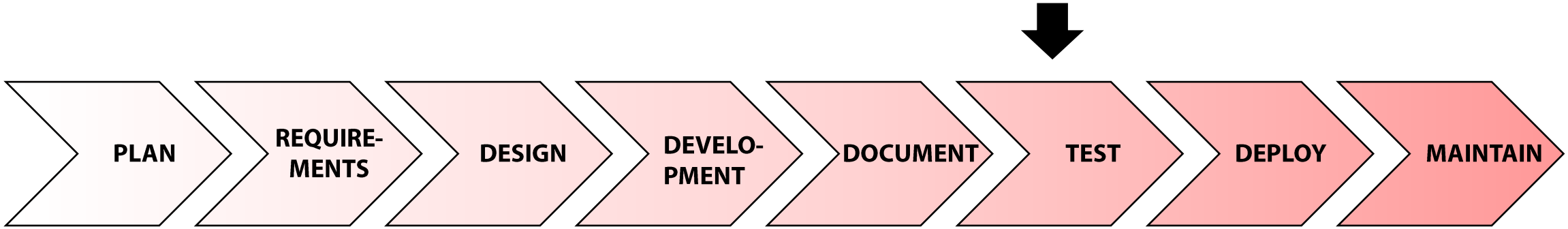
Software Development Lifecycle

- **Advantages**
  - Increased visibility of all relevant stakeholders into the development process
  - Efficient estimating, planning and scheduling
  - Improved risk management and cost estimation
  - Systematic software provision and improvement of customer satisfaction



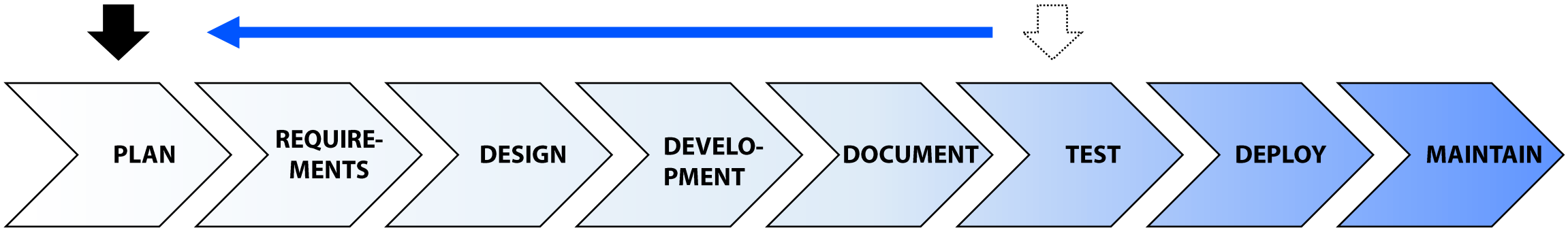
Software Development Lifecycle

- **Problems of traditional SDL**
  - Security related activities are deferred until the testing phase!!
    - Late in the SDLC after most of the critical design and implementation has been completed
    - Limited to code scanning and penetration testing
      - Might not reveal more complex security issues



Secure Software Development Lifecycle

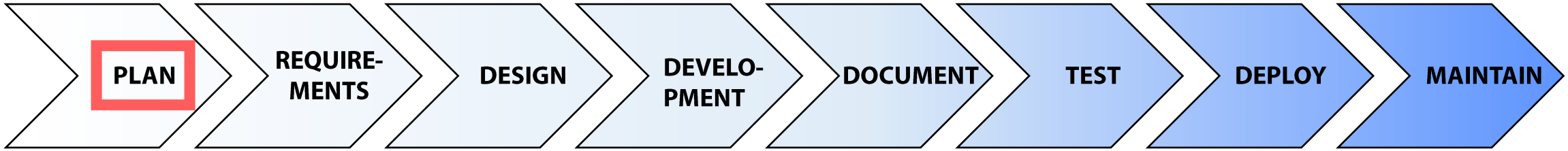
- **Effective security processes requires teams to "shift left"**
  - Starting at project inception and running throughout the project



https://www.redhat.com/en/topics/security/software-development-lifecycle-security

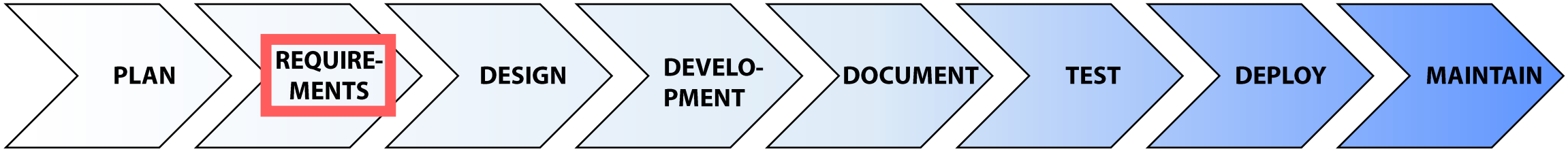
Secure Software Development Lifecycle

- **Plan**
  - Assess risks and security threat landscape
  - Evaluate the potential impact of security incidents
    - E.g., reputational risk to the business



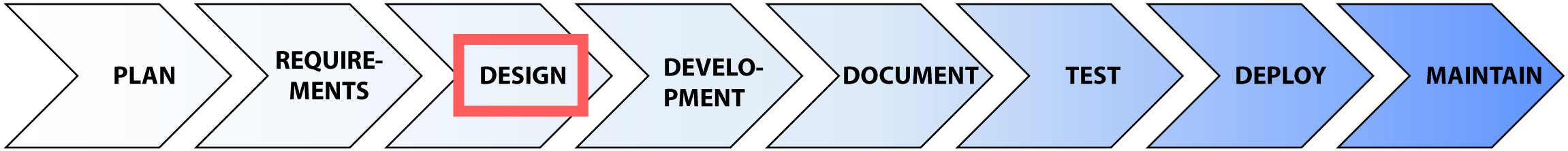
Secure Software Development Lifecycle

- **Requirements**
  - Include security requirements as part of defining functional requirements
    - E.g., must include vulnerability verification



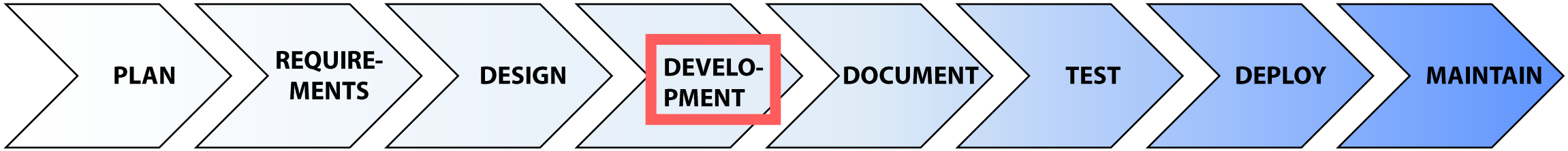
Secure Software Development Lifecycle

- **Design**
  - Make security considerations an integral part of the architecture plan
  - Evaluate security impact of design phase choices such as platform and UI



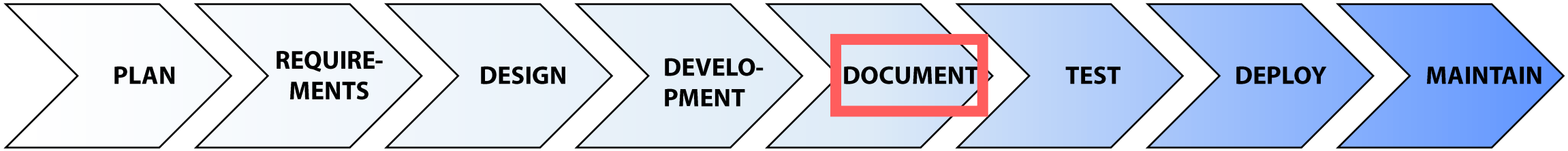
Secure Software Development Lifecycle

- **Development**
  - Educate developers on secure coding practices
  - Incorporate security testing tools in development process
    - E.g., static and dynamic analysis tools
  - Evaluate software dependencies and mitigate potential security risks



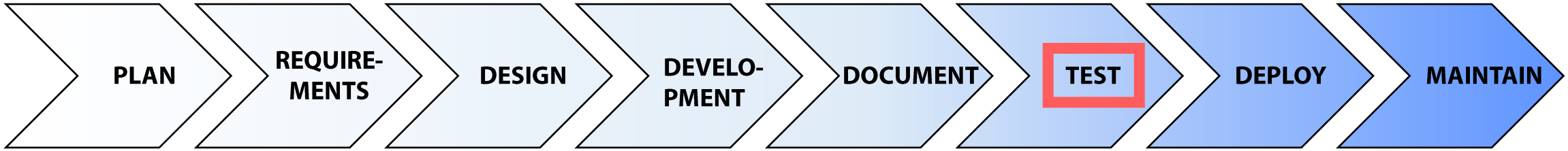
Secure Software Development Lifecycle

- **Document**
  - Security controls and process documentation
  - Assemble the information to prepare for audits, compliance checks, and security reviews



Secure Software Development Lifecycle

- **Test**
  - Implement code review processes
  - Perform security testing
    - E.g., static analysis and interactive application security testing



Static / dynamic analysis

- **Static analysis**
  - Examining source code without executing it
    - To identify potential security vulnerabilities
  - Also called whitebox testing
- **Dynamic analysis**
  - Running the program and analyzing its behavior during execution
    - To identify potential security vulnerabilities
  - Also called blackbox testing

Static analysis

- **Static analysis: symbolic execution**
  - A bug finding technique that is easy to use
  - Key idea
    - Evaluate the program on **symbolic input values** (rather than using a concrete input)
    - Use an automated theorem prover to check whether there are corresponding concrete input values that make the program fail

Static analysis

- **Static analysis: symbolic execution**

```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```

Static analysis

$x \mapsto A$
$y \mapsto B$

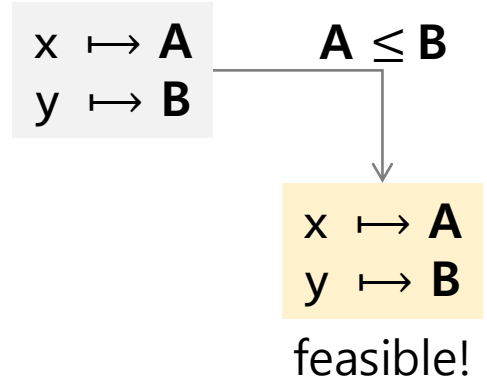
- **Static analysis: symbolic execution**

```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```

Static analysis

- **Static analysis: symbolic execution**

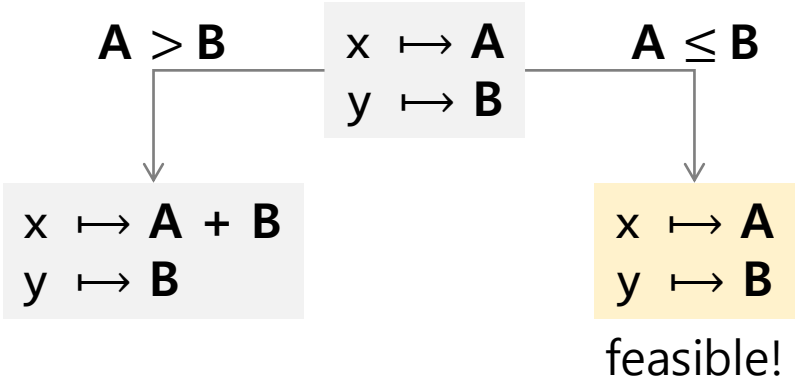
```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

- **Static analysis: symbolic execution**

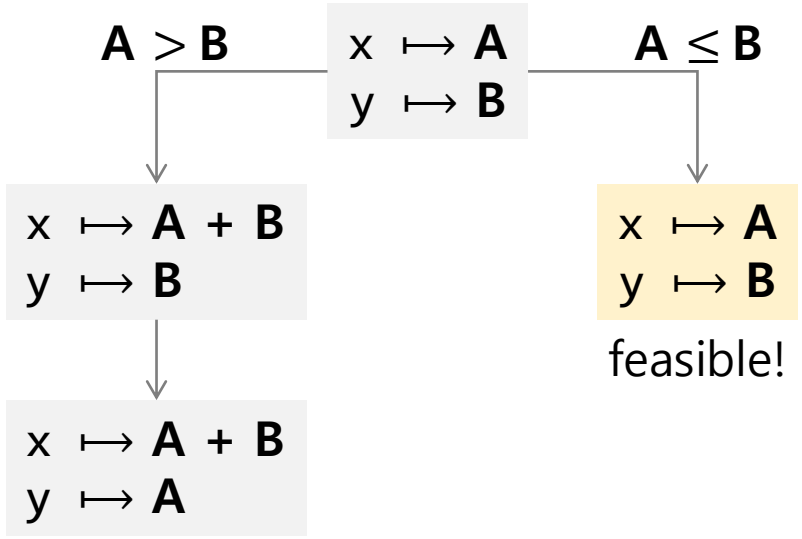
```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

- **Static analysis: symbolic execution**

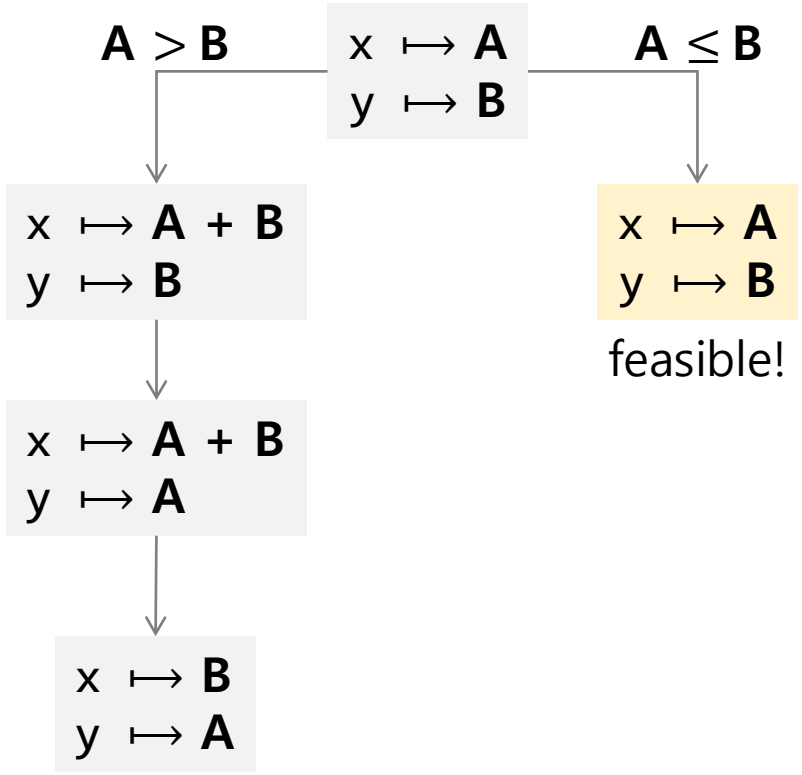
```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

- **Static analysis: symbolic execution**

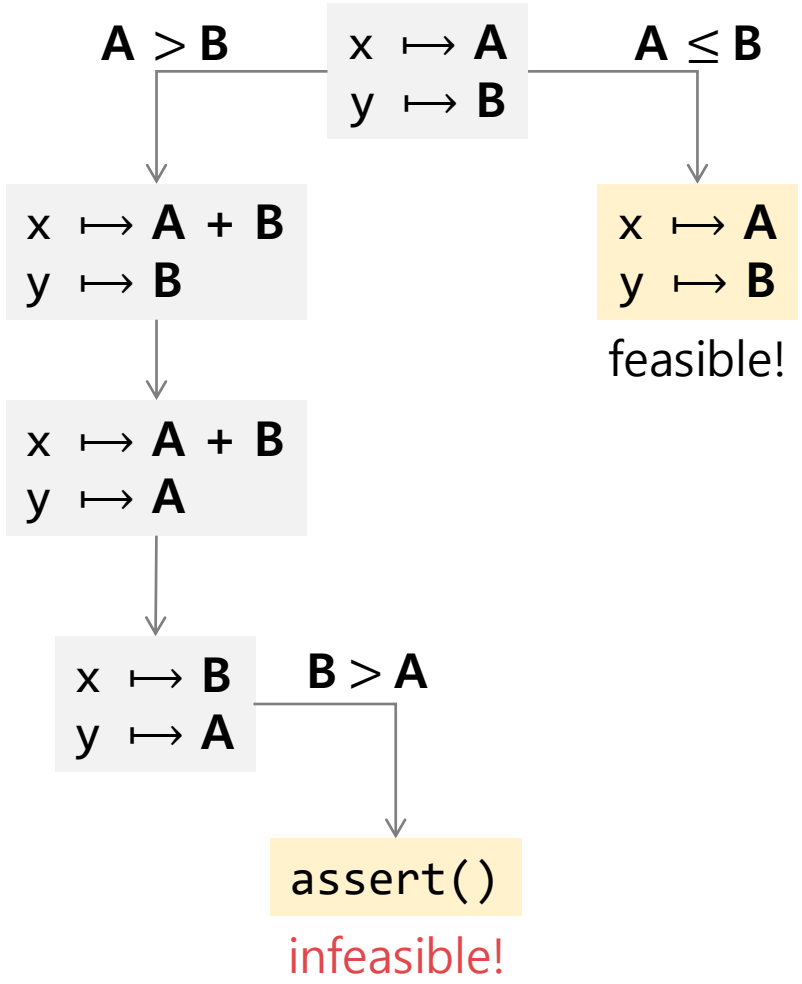
```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

- **Static analysis: symbolic execution**

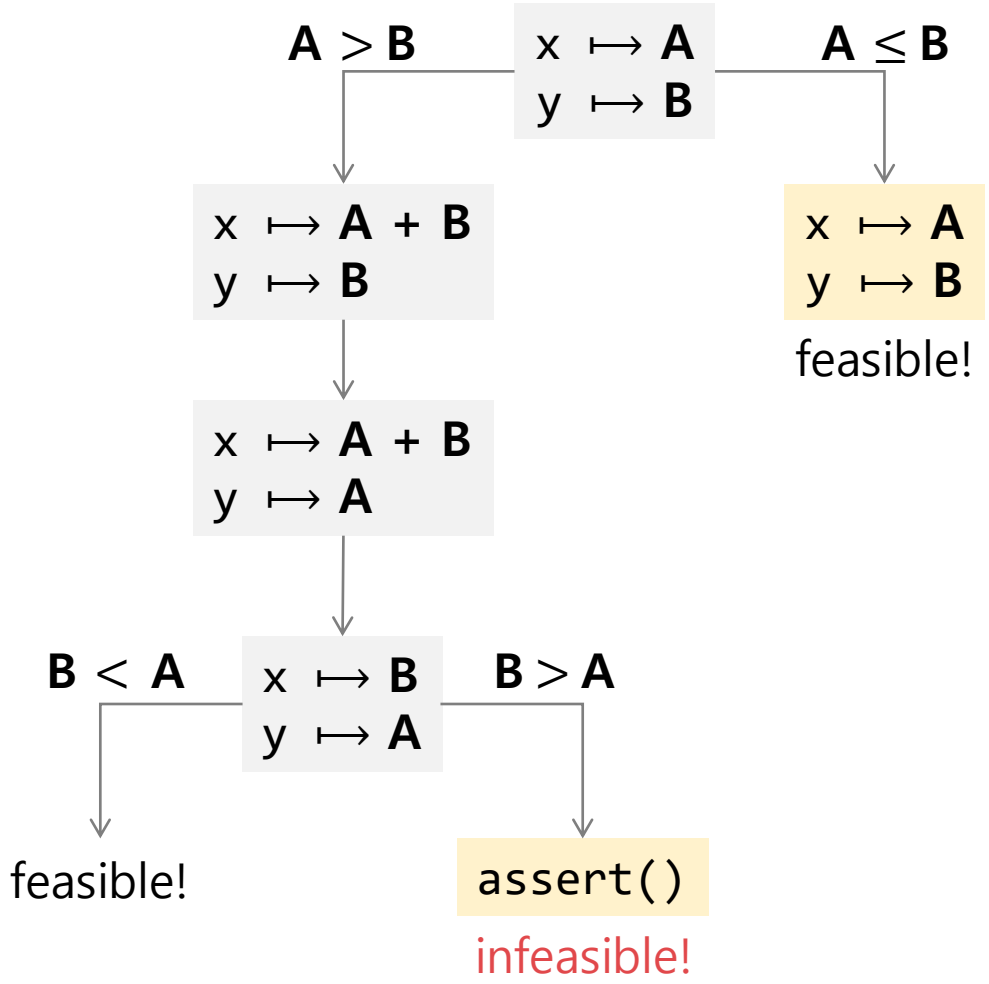
```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

- **Static analysis: symbolic execution**

```
1  def f (x, y):
2      if (x > y):
3          x = x + y
4          y = x - y
5          x = x - y
6          if (x - y > 0):
7              assert()
8      return (x, y)
```



Static analysis

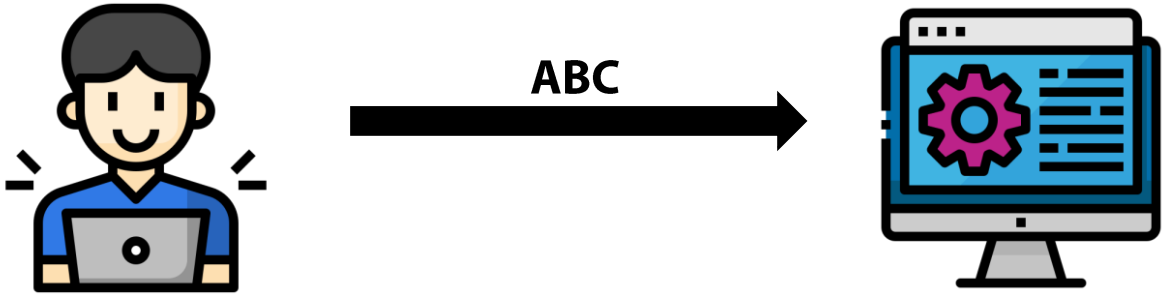
- **Static analysis: symbolic execution**
  - Explores all execution paths of a program and generates test input values
    - This helps uncover bugs sensitive to specific conditions or boundary conditions
  - Ensures high coverage of a program's execution paths
  - Can discover bugs in a program by exploring various execution paths
    - E.g., assertion violations, buffer overflows

Dynamic analysis

- **Dynamic analysis: fuzz testing (fuzzing)**
  - A technique used in security testing to discover bugs or vulnerabilities that may exist in software or systems
  - Key idea
    - Generating random input values to explore various parts of a system

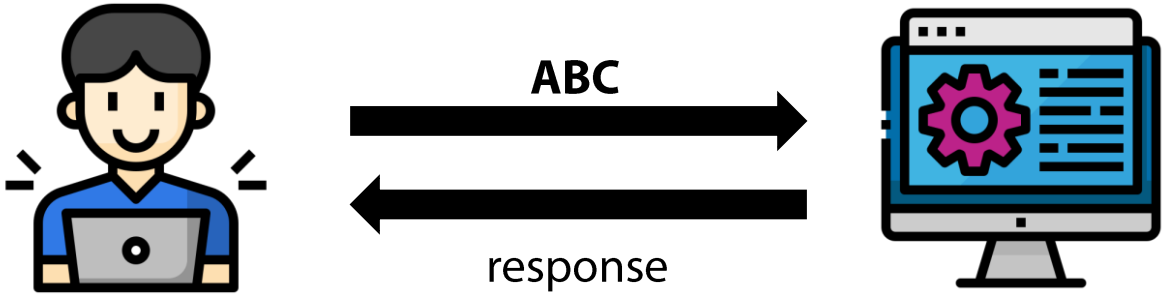
Dynamic analysis

- **Dynamic analysis: fuzz testing (fuzzing)**



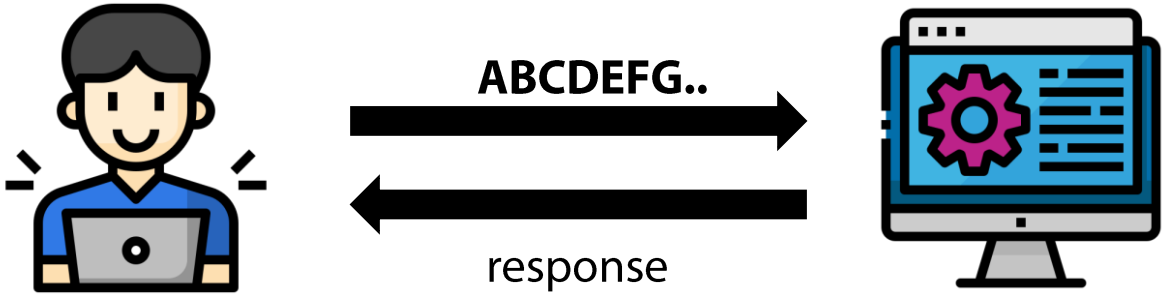
Dynamic analysis

- **Dynamic analysis: fuzz testing (fuzzing)**



Dynamic analysis

- **Dynamic analysis: fuzz testing (fuzzing)**

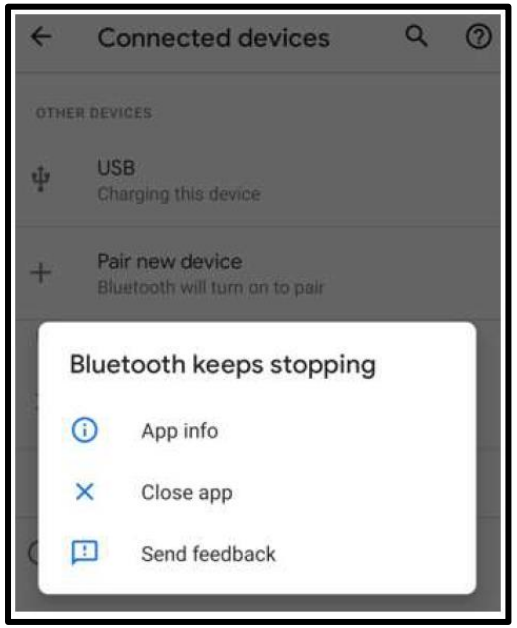


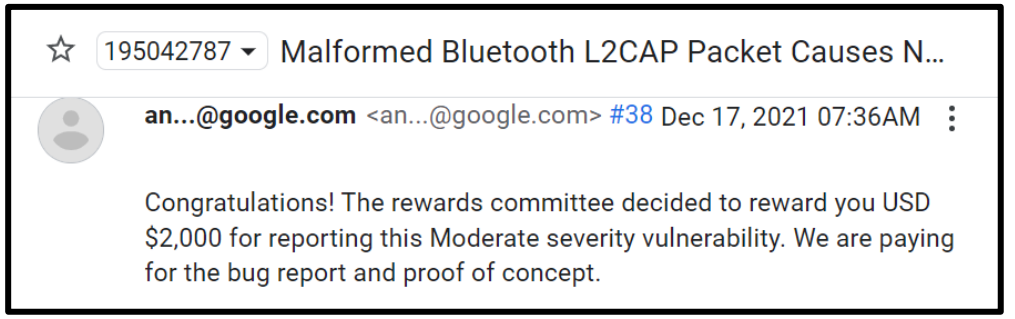
Dynamic analysis

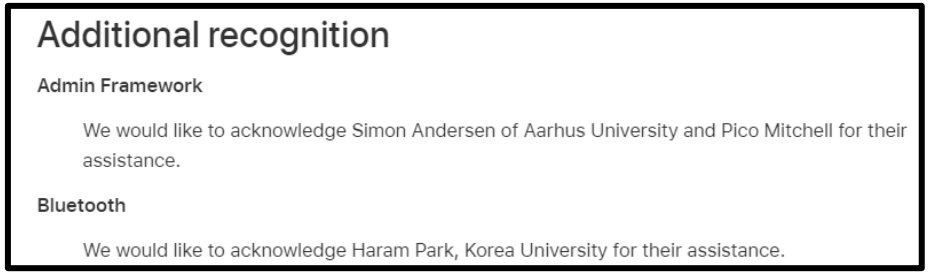
- **Dynamic analysis: fuzz testing (fuzzing)**
  - Typically automated using specialized tools and plays a crucial role in enhancing the stability and security of software
  - Particularly effective in finding security vulnerabilities
    - It can be run continuously until vulnerabilities are discovered and addressed

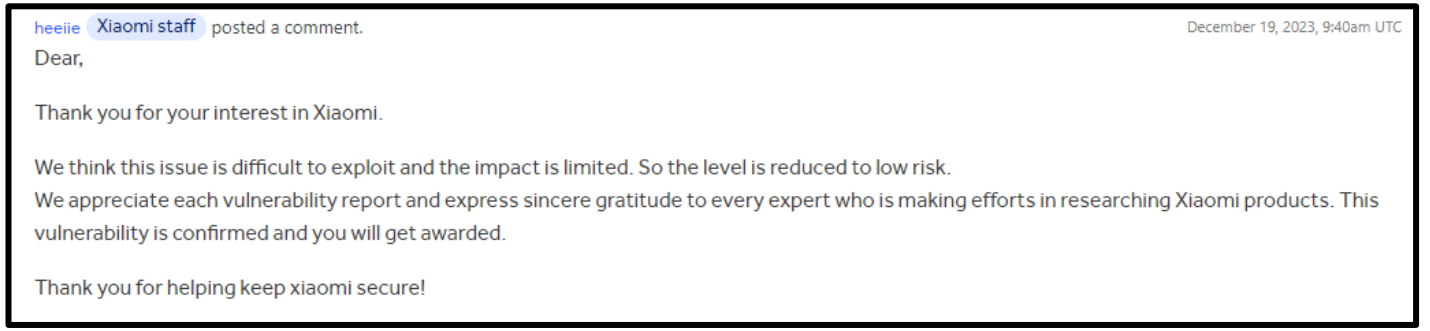
Dynamic analysis

- **Dynamic analysis: fuzz testing**









Static & dynamic analysis

- **Taint analysis**
  - To track the flow of sensitive or untrusted data through a program
  - The term "taint" refers to data that originates from an untrusted or potentially dangerous source, such as user input or external network communication
  - Taint analysis helps identify potential security vulnerabilities by tracing how tainted data propagates through the program and interacts with other data

Static & dynamic analysis

- **Taint analysis**

```
1  x = get_input();
2  y = 1;
3  z = x;
4  w = y + z;
5  print(w);
```

Static & dynamic analysis

- **Taint analysis**

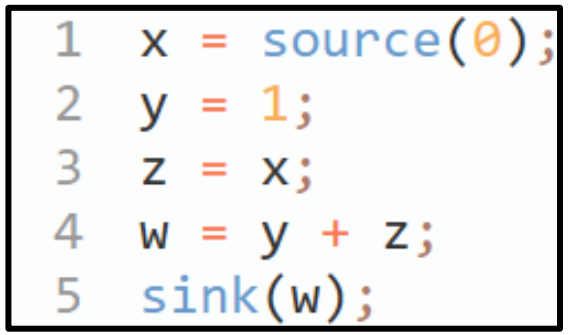
source();
- Indicates the point at which sensitive data enters the program
- This refers to data that originates outside the program
  - E.g., user input, external API calls, or reading data from the file system

```
1  x = source(0);
2  y = 1;
3  z = x;
4  w = y + z;
5  sink(w);
```

sink();
- Indicates the point within a program where a specific task is performed
- An area where sensitive data can reach and exploit that data to create security vulnerabilities

Static & dynamic analysis

- **Taint analysis**



x ⟶ 0->T(Tainted)

Static & dynamic analysis

- **Taint analysis**

```
1  x = source(0);
2  y = 1;
3  z = x;
4  w = y + z;
5  sink(w);
```

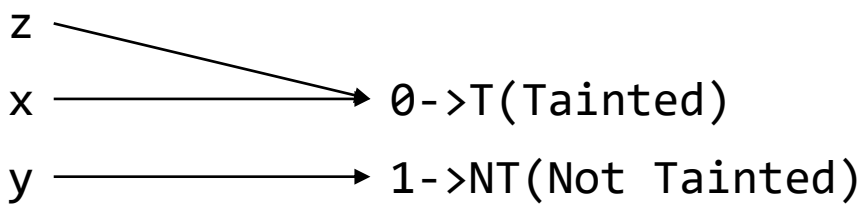
x ⟶ 0->T(Tainted)

y ⟶ 1->NT(Not Tainted)

Static & dynamic analysis

- **Taint analysis**

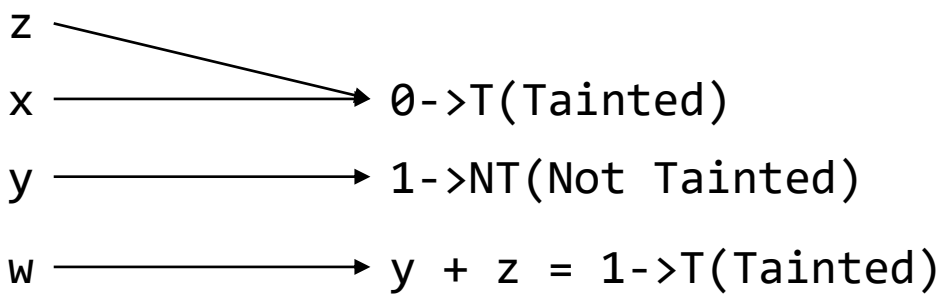
```
1  x = source(0);
2  y = 1;
3  z = x;
4  w = y + z;
5  sink(w);
```



Static & dynamic analysis

- **Taint analysis**

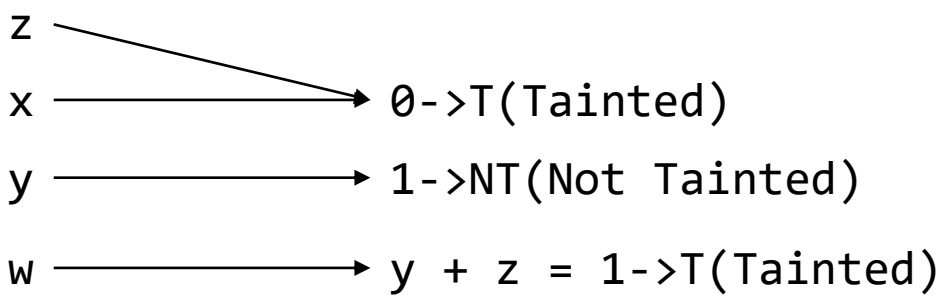
```
1  x = source(0);
2  y = 1;
3  z = x;
4  w = y + z;
5  sink(w);
```



Static & dynamic analysis

- **Taint analysis**

```
1  x = source(0);
2  y = 1;
3  z = x;
4  w = y + z;
5  sink(w);
```



Leak in the program!

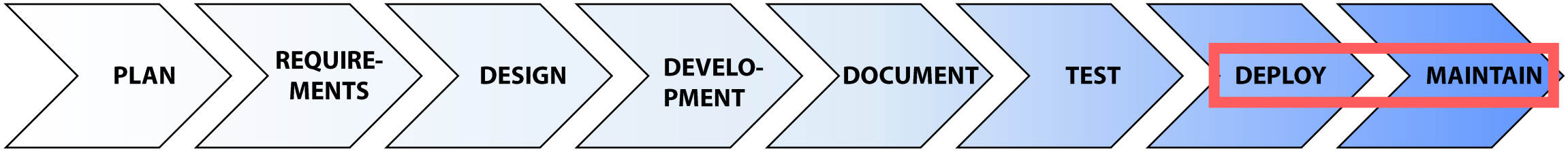
Static & dynamic analysis

- **Taint analysis**

```
1  uinput = input();
2  size   = uinput * 100;
3  ...
4  buf[100];
5  ...
6  print(buf[idx]);
```

Secure Software Development Lifecycle

- **Deploy**
  - Security assessment of deployment environment
  - Review configurations for security
- **Maintain**
  - Implement monitoring to detect threats
  - Be prepared to respond to vulnerabilities and intrusions with remediations



Next Lecture

- **Holiday!**
  - Don't forget to vote!