CALIFORNIA STATE UNIVERSITY
LONG BEACH
College of Engineering

CECS 303: Networks and Network Security

Common Network Defenses (cont'd)

Chris Samayoa

Week 14 – 2nd Lecture
4/21/2022

Course Information

CALIFORNIA STATE UNIVERSITY
LONG BEACH
College of Engineering

- CECS 303
  - Networks and Network Security – 3.0 units
- Class meeting schedule
  - TuTH 5:00PM to 7:15PM
  - Lecture Room: VEC 402
  - Lab Room: ECS 413
- Class communication
  - chris.samayoa@csulb.edu
  - Cell: 562-706-2196
- Office hours
  - Thursdays 4pm-5pm (VEC-404)
  - Other times by appointment only

Objectives

- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

Common Network Defenses

- Anti-virus
  - Signature-based and NGAV
- Firewalls
  - Host-based
  - Network based
    - Including NGFW(e.g. pfSense, OPNSense, NG Firewall)
    - Deep Packet Inspection (DPI)
- Intrusion Detection Systems
  - Can include network monitors
  - e.g. Snort
- Content Delivery Networks
  - Internet edge protection
  - e.g. Akamai (Kona DDoS Defender), Cloudflare

Common Network Defenses

- Large Cloud Providers
  - Amazon Web Services (AWS), Microsoft Azure, Google Cloud
- Network Security Focused Database Development
  - Parameterized database queries
  - Validate user-supplied data
- Software Development Testing
  - Dynamic Testing
  - Fuzzing
- Wireless Protection Best Practices
  - Strong passwords
  - VPNs
  - PKI

Objectives

- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

Content Delivery Networks

- Overview
  - Initially, a content delivery network (CDN) was used to speed up delivery of web content by caching web pages, images, video, etc.
    - Relieved web congestion by bringing content closer to providers
    - Focus is on distributing content of "origin" servers to local caches
  - Adds resiliency to web applications
    - Content is distributed geographically so that there is not a single point of failure
  - Often used to protect against DDOS attacks
    - Large availability of bandwidth and servers can typically withstand these attacks without affecting the web application
    - Avoids bringing down individual company networks (e.g. internet connections)
  - CDNs carried 56% of all internet traffic in 2017
    - Expected to carry 72% of internet traffic by 2022 according to Cisco

CDNs (cont'd)

- Overview of services
  - Increases performance in serving content by using caches
  - Protects against other common attacks
    - SQL injection
    - Cross-site scripting
  - Provides threat intelligence based on vast networks
    - Data can be used to protect customers from zero-day attacks
  - Attacker can still directly attack an organization's public IP addresses
  - Can be used as a proxy for filtering outbound user traffic for an organization

Objectives

- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

Cloud Providers

- Overview
  - Major cloud providers offer “built-in” protections against attacks including DDOS and malware
    - Some features are free and some are billed
  - Standard levels of DDOS protection are typically offered for free
    - Massive cloud infrastructures can be leveraged to easily defeat most DDOS attacks
  - Most other types of network protections are also available as additional features
    - Antivirus
    - Host / Network firewalls
    - IDS / Monitoring

Objectives

- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

DB Development

- Overview
  - SQL injection attacks can largely be avoided by using best practice development, including the following:
    - Parameterized database queries (Prepared Statements)
      - Bound / typed parameters
    - Parameterized stored procedures in the database
  - Important to keep network security in mind during all stages of development
- Additional recommendations
  - Keep software components patched
    - e.g. libraries, frameworks, web server software, etc.
  - Use appropriate service accounts for transactions from web servers to databases
    - Consider principle of least privilege when creating accounts
    - Never use root / admin accounts for services
    - Use separate service accounts for different web applications
  - Validate user input for expected format
  - Ensure that database error messages are not sent to client web browser

DB Development (cont'd)

- Unsafe SQL query example:

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
            + request.getParameter("customerName");
try {
      Statement statement = connection.createStatement( ... );
      ResultSet results = statement.executeQuery( query );
}
...
```

* Unvalidated "customerName" parameter is appended to the query

Source: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

DB Development (cont'd)

- Prepared Statements
  - Definition: precompiled SQL statement
  - Includes the use of variable binding
    - Meaning that user-supplied inputs can only used as a variable and not separate commands
  - Developer must first define all SQL code
    - Each parameter is passed to SQL query later on
    - Allows database to distinguish between code and data
  - Allows database to distinguish between code and data
  - Attacker unable to change intent of query
    - Even when using SQL commands within user-supplied input
    - e.g. userID of ***tom' or '1' = '1*** would be interpreted literally as the name of a user

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

DB Development (cont'd)

- Prepared statement example using Java:

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

  - '?' is a placeholder for the variables
  - 'pstmt.setString' command passes the prepared statement the user-supplied input
  - SQL injection attack has now been avoided

Source: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

DB Development (cont'd)

- Prepared statement example 2 using Java:

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
// FIXME: do your own validation to detect attacks
String query = "SELECT id, firstname, lastname FROM authors WHERE firstname = ? and lastname = ?";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
    ResultSet results = pstmt.execute( );
}
```

  - '?' is a placeholder for the variables
  - 'pstmt.setString' command passes the prepared statement the user-supplied input
    - Numbered from left to right for placeholder values
  - SQL injection attack has now been avoided

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

DB Development (cont'd)

- Stored Procedures
  - Definition: saved SQL code that can be reused
    - Stored within database
    - Can be reused by different clients sending input parameters
  - Also vulnerable to SQL injection attacks
  - Use of dynamic SQL queries should be avoided in stored procedures
    - Not the norm to use dynamic SQL in stored procedures (but possible)

DB Development (cont'd)

- Stored procedure example using Java:

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
  CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
  cs.setString(1, custname);
  ResultSet results = cs.executeQuery();
  // … result set handling
} catch (SQLException se) {
  // … logging and error handling
}
```

  - 'sp_getAccountBalance' is the stored procedure with the same query as previous example (1)

Objectives

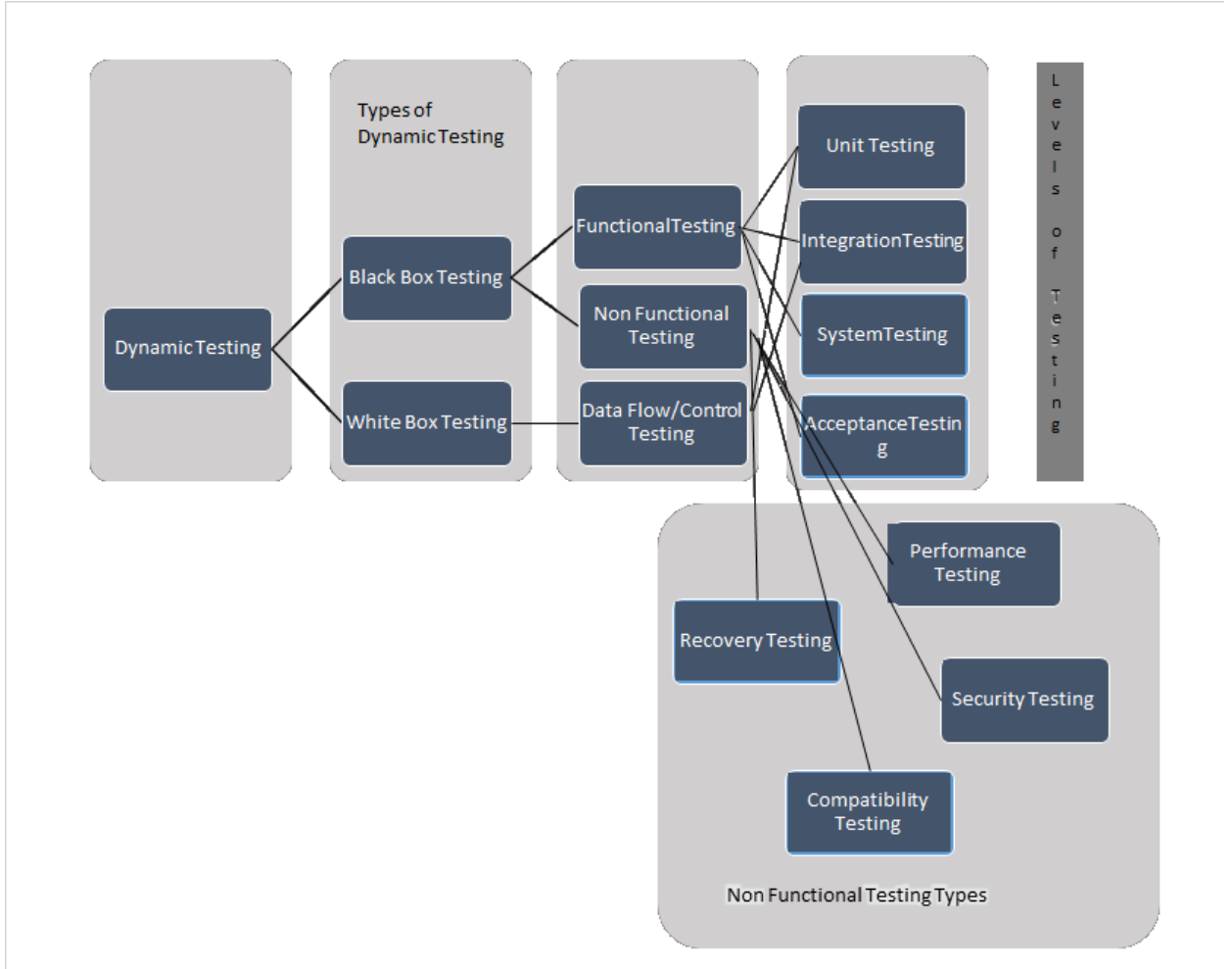
- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

Dynamic Testing

- What is it?
  - Testing of software behavior with dynamic variables to find weak areas in software
    - Essentially software validation
    - Attempt to test in “real” environment
  - Code is executed
    - Hopefully with extremes of valid/incalid input
    - Altered timing – time of check to time of use (TOCTOU)
    - Stress tests
  - Typical of QA testing for a product
- Types of dynamic testing
  - White Box
    - Internal structure / design is known to tester
    - Typically performed by developers
    - Goal is to check system performance
  - Black Box
    - Internal structure / design is not known to the tester
    - Typically performed by third party or QA team
    - Goal is to ensure functionality of system and to test non-functional items as well such as security
      - e.g. verify that authentication / authorization is functioning properly

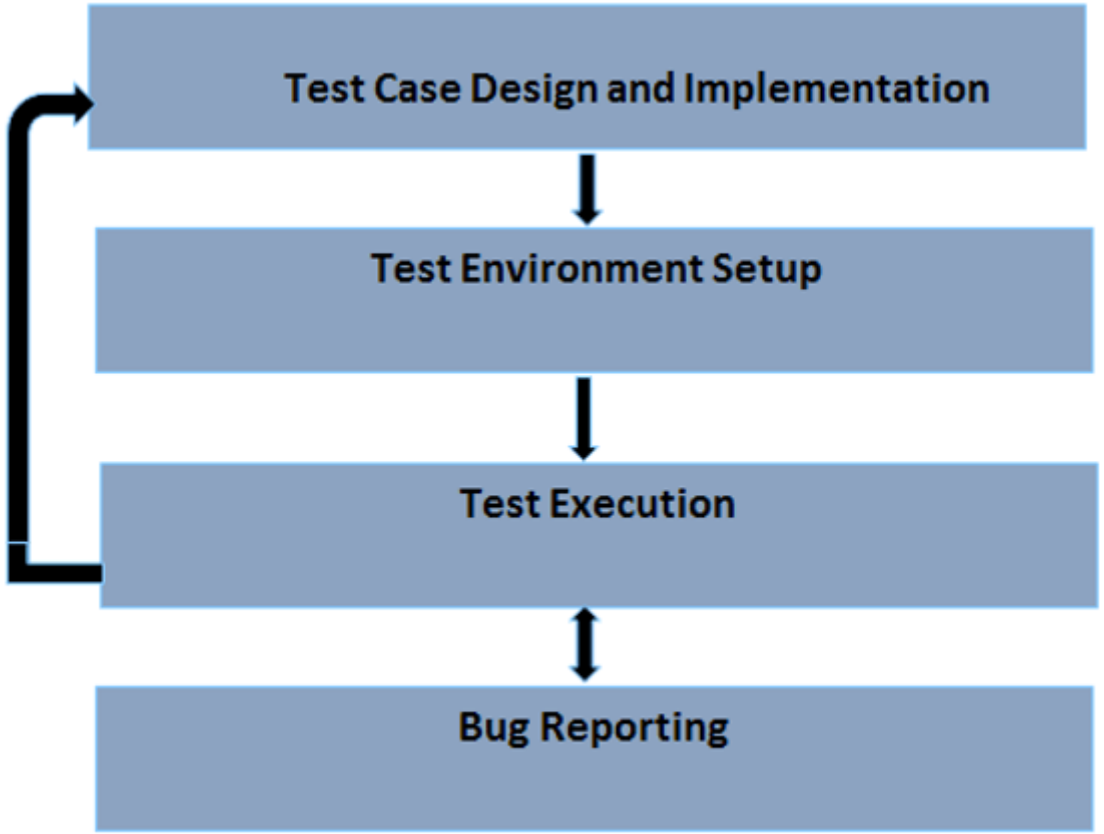
Source: https://www.guru99.com/dynamic-testing.html

Dynamic Testing



Source: https://www.guru99.com/dynamic-testing.html

Dynamic Testing

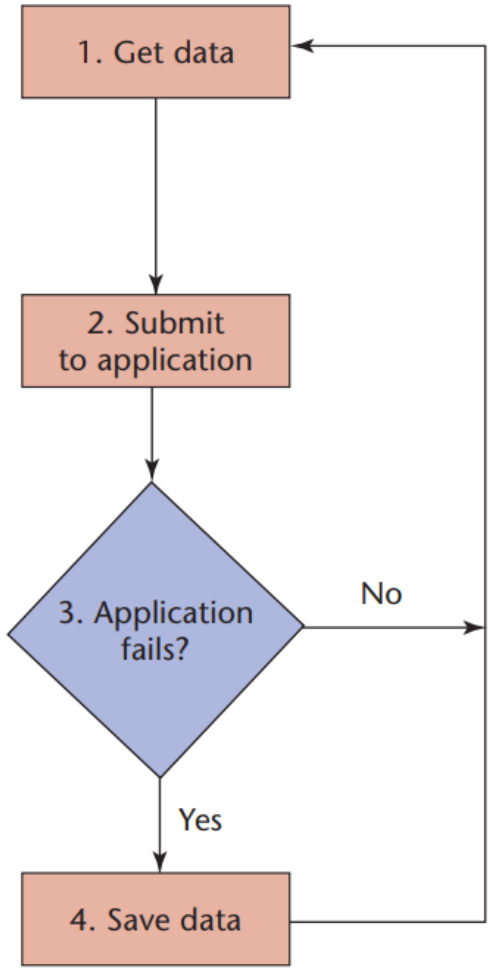


STLC (Software Testing Lifecycle)

Source: https://www.guru99.com/dynamic-testing.html

Fuzzing

- What is it?
  - Highly automated testing technique that covers a multitude of boundary cases using invalid data as input in search of exploitable vulnerabilities
    - ➢ Attempt to ensure that a product does not do what it is not supposed to do
    - ➢ Behavior of application is observed for unexpected results / flaws
      - ○ e.g. crashes or memory leaks
  - Random testing
    - ➢ Use static / dynamic testing results as starting points
  - Tool / technique used by both testers and potential attackers
- Types of inputs typically used
  - Extreme limits / bounds (or beyond)
    - ➢ Value, size type, etc.
  - Test via different avenues
    - ➢ e.g. command line and GUI



Fuzzing (cont'd)

- Types
  - Black Box
    - Tool knows nothing about the program or expected input
    - Easy to use, but will likely only have shallow findings
  - Grammar Based
    - Generate input informed by grammar
    - More work to configure and use, but should be able to dive deeper in the software
  - White Box
    - Tool generates inputs informed by the code of the target software
    - Computationally expensive because it involves program analysis
- Inputs
  - Mutation
    - Take an expected input and modify / mutate it (e.g. by flipping bits)
    - Expected input can be human-produced or automated
      - Can force mutation to follow grammar
  - Generation-based
    - Generate input from scratch
    - Based on models of input (typically grammar)
  - Combinations
    - Manually create initial input, mutate, and then generate new inputs

File-based Fuzzing

- Mutate or generate inputs and run the target program with them
- Example for grammar-based file
  - Specified as regular expressions or CFGs (content free grammars)
    - ➢ e.g. Blab tool

```
% blab -e '(([wrstp][aeiouy]{1,2}){1,4} 32){5} 10'
soty wypisi tisyro to patu
```

Network-based Fuzzing

- Fuzzer acts as ½ of communicating pair
  - Potential inputs could come from replaying previous interaction mutating inputs or by using generational inputs
- Functions as “man-in-the-middle” for purposes of testing
  - Fuzzer can mutate inputs as they are exchanged by two other parties
- Applications
  - SPIKE
  - BURP Intruder

Fuzzing (cont'd)

- Limits
  - Random sample of behavior
  - Usually only finds simple flaws
  - Often a rough measure of software quality
  - Not a proof that software is correct
    - Bugs found are not a comprehensive list
- Goals
  - Find root causes of issues found
    - Commonalities among inputs that cause crashes or unexpected behavior can point towards the same bug
  - Determine whether the unexpected behavior is an exploitable vulnerability
    - e.g. buffer overrun

Objectives

- Common Defenses – Overview
  - Content Delivery Networks
  - Large Cloud Providers
  - Database Development
  - Software Testing
  - Wireless Protection

Wireless Protection

- Strong passwords
  - Make it more difficult for your wireless network password to be brute-forced
  - WEP vs WPA2
    - WPA3 available but many devices cannot yet use it
- VPNs
  - Keep data protected from potentially malicious wireless access points (e.g. airports, coffee shops, etc.)
- PKI
  - Certificates and certificate authorities (CA) to be discussed next lecture