

Software Security

CS436/536/636

April 20, 2009

Gary Warner

Goal

Let's NOT fix the same problem 1,000 times.

Let's LEARN from the problems we fixed, and prevent them from happening again.

Example

Zlib - a corrupt PNG file can give full control – March 2002

IE - a corrupt PNG file can give full control – Nov 2002

libPNG – a corrupt PNG file can Denial of Service (crash) a browser – Aug 2004

MS GDI+ - a corrupt JPG file can give full control – Sep 04

Zlib – a corrupt PNG file can give full control – Jul 05

MITRE

Top 25 Coding Weaknesses

MITRE identified 700 common “Coding Weaknesses” and made a Top 25 list, placing the results in three categories

<http://cwe.mitre.org/top25/>

Category 1: Insecure Interaction Between Components

Improper input validation

Improper encoding or escaping of output

Failure to preserve SQL query structure (aka "SQL injection")

Failure to preserve Web page structure (aka "cross-site scripting")

Failure to preserve operating system command structure (aka "OS command injection")

Cleartext transmission of sensitive information

Cross-site request forgery (CSRF)

Race condition

Error message information leak

Category 2: Risky Resource Management

Failure to constrain operations within the bounds of a memory buffer

External control of critical state data

External control of file name or path

Untrusted search path

Failure to control generation of code (aka "code injection")

Download of code without integrity check

Improper resource shutdown or release

Improper initialization

Incorrect calculation

Category 3: Porous Defenses

Improper access control (authorization)

Use of a broken or risky cryptographic algorithm

Hard-coded password

Insecure permission assignment for critical resource

Use of insufficiently random values

Execution with unnecessary privileges

Client-side enforcement of server-side security

OWASP

Open Web Application Security Project

Examples and outline drawn heavily from
“Secure Programming with Static Analysis” by
Brian Chess and Jacob West

Generic vs. Context Specific

Generic Defects are coding issues that do not rely on what kind of program we are writing – for example, a buffer overflow

Context-specific Defects rely on the data being evaluated, for example, a Credit Card processing error

Seven Pernicious Kingdoms

1. Input Validation and Representation
2. API Abuse
3. Security Features
4. Time and State
5. Error Handling
6. Code Quality
7. Encapsulation

1. Input Validation

Mis-handled user input leads to trouble:

- Unvalidated input
- Buffer overflows
- XSS (Cross Site Scripting)
- SQL Injection

Fuzzing

Michael Sutton, Adam Greene and Pedram Amini book – “Fuzzing: Brute Force Vulnerability Discovery”

1. Identify target
2. Identify inputs
3. Generate fuzzed data
4. Execute fuzzed data
5. Monitor for exceptions
6. Determine exploitability

Variant Input

Format strings . . .

Character translation . . .

Directory traversal

Command line injection

Online

Targets

- Web mail, Discussion boards, Wikis, Blogs, ERP systems,

Attacks

- Denial of Service, XSS, SQL injection, Directory Traversal, Weak authentication, Weak session management, Buffer overflow, Remote command execution, Remote file inclusion, Vulnerable libraries

Want to play?

<http://zero.webappsecurity.com/>

<http://testphp.acunetix.com/>

<http://demo.testfire.net/>

<http://www.owasp.org/> (look for WebGoat)

2. API Abuse

Attacks made by calling a “more privileged” function.

Example – lpr, the linux Line Printer command came installed with “setuid root” to allow it to configure any printer it could see.

lpr had an “open()” and “access()” subroutine to be able to print any user’s files.

We’ll look more at that in a minute . . .

3. Security Features

These are application areas of authentication, access control, confidentiality, cryptography, and privilege management

- Broken Access Control
- Broken Authentication and Session Management
- Insecure Storage

4. Time & State

Programs are written by programmers who assume linear execution will be performed by a single thread on a single user machine.

Those programs may actually execute in shared memory on a computer being used by THOUSANDS of users!

- Race conditions
- XSS Java – (using “source includes” to bypass the “Same Origin Policy” rules of Javascript)

A Java “Web services” XSS

Let’s say that a certain Web services app is used to look up your data in a sales leads database.

Get /object.json (JavaScript Object Notation)

Host: www.example.com

Cookie: JSESSIONID=F2rN6HopNzsfXFj...SbAiTnRR

The data comes back . . . {fname: Brian, lname: Chess,
phone: 6502135600, purchases: 60000.00, email:
brian@fortifysoftware.com}

A Java “web services” XSS

Now what happens when another piece of Javascript is run on a server.

This code has code to call data and send it back via email to the attacker. Part of this involves creating a “local version” of the “captureObject” Javascript function.

Within the script, there is a tag:

```
<script src=http://www.example.com/object.json>
```

What happens? The additional data is included . . . Because the user’s browser has a valid cookie for “example.com”, it properly authenticates and returns the data, and because there was a “local” version of “captureObject”, the example.com JSON call will use the “local” version instead of the “system” version. This makes the data “local” and thus not in violation of the “Same Origin” rule.

5. Error Handling

What do we do when we crash? What do we do when we receive invalid input? Do we “die gracefully?” or do we leave the system in an unpredictable or exploitable state?

6. Code Quality

Poor quality leads to unpredictable behavior. A user of the system may use this code to stress the system of execution in unexpected ways. Can we create an infinite loop? A Denial of Service?

7. Encapsulation

Drawing strong boundaries. For instance, preventing your Web browser mobile code from accessing data from other mobile code.

Preventing users from seeing each other's data.

Differentiating “validated” and “unvalidated” data.

* Environment

Source code is often a victim of the environment in which it is executed.

Compiler flags

Configuration files

File permissions

Network settings

