the Availability Digest

www.availabilitydigest.com

The 25 Most Exploitable Programming Errors

February 2013

The Department of Homeland Security (DHS) Office of Cybersecurity and Communications publishes a detailed list of the twenty-five most egregious programming errors that lead to exploitable security vulnerabilities that have made software so vulnerable to hackers and cybercriminals. International in scope and free for public use, the *Common Weakness Enumeration (CWE)* (https://buildsecurityin.us-cert.gov/swa/cwe/) is a community-developed dictionary of software weaknesses. The top twenty-five CWEs represent the most significant exploitable software constructs in this dictionary

The CWE provides descriptions of the most common and serious exploitable software constructs. They are often easy to find and to exploit by cybercriminals. They are dangerous because they will frequently allow attackers to completely take over the system, steal data, or prevent the system from working at all. Consequently, the CWE can aid in the education and training of programmers on how to eliminate all-too-common errors that can be compromised by malware.

The CWE is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. MITRE maintains the CWE web site with the support of the U.S. Department of Homeland Security's National Cyber Security Division. The web site provides detailed descriptions of these errors along with authoritative guidance for mitigating and avoiding them. Furthermore, the CWE web site contains data on more than 800 programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

The CWE top twenty-five vulnerabilities are updated each year. The hundreds of vulnerabilities listed on the CWE web site are prioritized using inputs from over twenty organizations. Prioritization is based on prevalence, importance, and likelihood of exploit using the *Common Weakness Scoring System, CWSS* (http://cwe.mitre.org/cwss/).

The Top Twenty-Five Vulnerabilities

The most recent top twenty-five vulnerabilities listed on the CWE web site include the following in order of their seriousness:

Rank	**Vulnerability**
1	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4	Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')
5	Missing Authentication for Critical Function

6	Missing Authorization
7	Use of Hard-coded Credentials
8	Missing Encryption of Sensitive Data
9	Unrestricted Upload of File with Dangerous Type
10	Reliance on Untrusted Inputs in a Security Decision
11	Execution with Unnecessary Privileges
12	Cross-Site Request Forgery (CSRF)
13	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
14	Download of Code Without Integrity Check
15	Incorrect Authorization
16	Inclusion of Functionality from Untrusted Control Sphere
17	Incorrect Permission Assignment for Critical Resource
18	Use of Potentially Dangerous Function
19	Use of a Broken or Risky Cryptographic Algorithm
20	Incorrect Calculation of Buffer Size
21	Improper Restriction of Excessive Authentication Attempts
22	URL Redirection to Untrusted Site ('Open Redirect')
23	Uncontrolled Format String
24	Integer Overflow or Wraparound
25	Use of a One-Way Hash without a Salt

Types of Vulnerabilities

The top twenty-five vulnerabilities are further characterized into categories that describe the general form of the vulnerability that they pose:

Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

Rank	Vulnerability
1	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
4	Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')
9	Unrestricted Upload of File with Dangerous Type
12	Cross-Site Request Forgery (CSRF)
22	URL Redirection to Untrusted Site ('Open Redirect')

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

Rank	Vulnerability
3	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
13	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
14	Download of Code Without Integrity Check
16	Inclusion of Functionality from Untrusted Control Sphere
18	Use of Potentially Dangerous Function
20	Incorrect Calculation of Buffer Size
23	Uncontrolled Format String
24	Integer Overflow or Wraparound

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

Rank	Vulnerability
5	Missing Authentication for Critical Function
6	Missing Authorization
7	Use of Hard-coded Credentials
8	Missing Encryption of Sensitive Data
10	Reliance on Untrusted Inputs in a Security Decision
11	Execution with Unnecessary Privileges
15	Incorrect Authorization
17	Incorrect Permission Assignment for Critical Resource
19	Use of a Broken or Risky Cryptographic Algorithm
21	Improper Restriction of Excessive Authentication Attempts
25	Use of a One-Way Hash without a Salt

Detailed Vulnerability Description

For each vulnerability, a great deal of information is given.

Summary

An overall evaluation of the seriousness of the vulnerability is given. For instance, SQL Injection (Rank 1) is evaluated as follows:

Weakness Prevalence:	High
Remediation Cost:	Low
Attack Frequency:	Often
Consequences:	Security bypass, data loss
Ease of Detection:	Easy
Attacker Awareness:	High

Discussion

A general description is given of the effects of the vulnerability. For instance, for SQL Injection, the CWS points out that if attackers can influence the SQL statements that you use to access your database, they have access to all of your data. If you use SQL queries in security controls such as authentication, attackers can alter the logic of these queries to bypass security. SQL injection was responsible for the high-profile compromises of companies such as Sony, PBS, and MySQL.com.

Technical Details

The CWE goes into depth in its explanation of the vulnerability. The explanation includes a detailed technical description, the points in the project life cycle where the vulnerability might be introduced, and its common consequences.

Coding Examples

Extensive coding examples showing improper coding techniques that can be exploited by an attacker are provided. An illuminating example is given for SQL Injection. This example is copied directly from the CWE. Consider the following SQL statement:

```
string query = "SELECT * FROM items WHERE owner = "' + userName + "' AND itemname = "' +
ItemName.Text + "'";
```

The query that this code intends to execute is:

```
SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:

```
name' OR 'a'='a
```

for itemName, then the query becomes the following:

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

The addition of the:

```
OR 'a'='a'
```

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user. The query now returns all entries stored in the items table, regardless of their specified owner.

This is just one example of the multitude of coding examples for the top twenty-five vulnerabilities found in the CWE.

Detection Methods

Various methods for detecting the vulnerability are presented, including the effectiveness of automatic static analysis, automatic dynamic analysis, and manual analysis.

Prevention and Mitigation

Several suggestions are offered for preventing or mitigating the vulnerability. They include procedures to follow in the architectural, design, implementation, deployment, and operational phases of the project.

Summary

The CWE is another example of the Department of Homeland Security's fight against cybercrime. Last December, 2012, it issued a warning to disable Java 7 because of vulnerabilities that Oracle has yet been unable to correct.[1]

The detailed descriptions for the "CWE Top 25 Most Dangerous Software Errors" can be found at http://cwe.mitre.org/top25/#CWE-89.

[1] Department of Homeland Security Says, "Disable Java", *Availability Digest*; January 2013.
http://www.availabilitydigest.com/public_articles/0801/disable-java.pdf

A handy Pocket Guide (30 pages) summarizing the top twenty-five vulnerabilities, entitled "Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses," may be found at https://buildsecurityin.us-cert.gov/swa/downloads/KeyPracticesMWV13_02AM091111.pdf.