19 Deadly Sins of Software Security

M. Howard, D. LeBlanc, J. Viega

Security Defects

- We live in an age with constant threat of security breaches
  - Holes in web software
  - Flaws in server software
- Security defects very easy to make
  - Blaster worm defect only two lines long
  - One line error can be catastrophic
- Here we look at 19 common security defects (sins of security)

Sin 1 : Buffer Overruns

- You've heard this one many times…
- Occurs when a program allows input to write beyond the end of the allocated buffer
  - Program might crash or allow attacker to gain control
  - Still possible in languages like C#, Java since they use libraries written in C/C++ but more unlikely

Buffer Overflow Example

```
void DontDoThis(char *input)
{
        char buf[16];
        strcpy(buf, input);
        printf("%s\n", buf);
}

int main(int argc, char *argv[])
{
        DontDoThis(argv[1]);
        return 0;
}
```

Buffer Overflow

- Here's what caused the Morris finger worm

```
char buf[20];
gets(buf);
```

- What about this?

```
bool CopyStructs(InputFile *pInFile, unsigned long count)
{
        unsigned long i;
        m_pStructs = new Structs[count];
        for (i=0; i<count; i++)
        {
                if (!ReadFromFile(pInFile, &(m_pStructs[i])))
                        break;
        }
}
```

Buffer Overflow

- Or this?

```
#define MAX_BUF 256

void BadCode(char *input)
{
        short len;
        char buf[MAX_BUF];

        len = strlen(input);

        if (len < MAX_BUF)
                strcpy(buf, input);
}
```

Buffer Overflow

- Or this?

```
#define MAX_BUF 256

void BadCode(char *input)
{
        short len;
        char buf[MAX_BUF];

        len = strlen(input);

        if (len < MAX_BUF)
                strcpy(buf, input);
}
```

If a short is 2 bytes and input > 32767, then len becomes a negative number

If input is not null-terminated…

Slightly better: Use size_t to define size for MAX_BUF and len

Spotting Buffer Overflows

- Look for input read from the network, a file, the user interface, or the command line
- Transfer of data from input to internal structures
- Use of unsafe string handling calls
- Use of arithmetic to calculate an allocation size or remaining buffer size

Buffer Overrun Examples

- CVE = Common Vulnerabilities and Exposures List
- CVE-2008-0778
  - Multiple stack-based buffer overflows in an ActiveX control in QTPlugin.ocx for Apple QuickTime 7.4.1 and earlier allow remote attackers to cause a denial of service (crash) and possibly execute arbitrary code via long arguments to the (1) SetBgColor, (2) SetHREF, (3) SetMovieName, (4) SetTarget, and (5) SetMatrix methods.
- CVE-2003-0352
  - Buffer overflow in a DCOM interface for RPC in Windows … allows remote attackers to execute arbitrary code via a malformed message, as exploited by Blaster/MSblast/LovSAN/Nachi/Welcia worms

Buffer Overflow Redemption

- Replace dangerous string handling calls with safe ones
  - C++ STL String Library considered safe
  - Check loops and array accesses
  - Replace static arrays with STL Containers

Sin 2 : Format String Problems

- A C/C++ type of problem
- First mentioned June 23, 2000

```
void main(int argc, char * argv[])
{
        printf(argv[1]);
}
```

- Pretty simple, what could go wrong?

Format String

- What if the program is invoked as :

  bug.exe “%x %x”

- Output something like:

  12FFC0 4011E5

The %x specifier reads the stack
4 bytes at a time and outputs them

Leaks important info to the attacker

Format String

- Another obscure format string: %n

```
unsigned int bytes;
printf("%s%n\n", argv[1], &bytes);
printf("Input is %d characters long.\n", bytes);

Usage:

bug.exe "Hello"
Hello
Input is 5 characters long
```

The %n specifier writes 4 bytes at a time based on the length of the previous argument

Carefully crafted, allows an attacker to place own data into the stack

Examples

- CVE-2000-0573
  - The lreply function in wu-ftpd 2.6.0 and earlier does not properly cleanse an untrusted format string, which allows remote attackers to execute arbitrary commands.
  - "Providing *remote* root since 1994"
- CVE-2007-4708 TA07-352A
  - Format string vulnerability in Address Book in Apple Mac OS X 10.4.11 allows remote attackers to execute arbitrary code via the URL handler.

Redemption

- printf(“%s”, user_input);
- Or filter user input for dangerous characters

Sin 3 : Integer Overflows

- When an unsigned integer gets too big for the number of bits allocated, it overflows back to 0
  - For signed integers, positive numbers suddenly become negative numbers
- “Obvious” errors where integers are multiplied/added/etc. and overflow
  - Result can be very bad and unpredictable behavior if relational operators suddenly behave the opposite of how they are supposed to
- Also many less obvious errors

Casting

- Implicit type casting is a frequent cause of integer overflows
- Most languages require the same types to be compared so an up-cast is done

```
const long MAX_LEN = 0x7FFF;

short len = strlen(input);

if (len < MAX_LEN)
{
        // Do stuff
}
```

If a short is 2 bytes and input > 32767, then len becomes a negative number

Casting

- Signed int to Larger signed int
  - Smaller value is sign-extended
    - 0x7F to an int becomes 0x0000007F
    - 0x80 to an int becomes 0xFFFFFF80
- Signed int to Larger unsigned int
  - Positive numbers behave as expected
  - Negatives unexpected
    - (char) -1 becomes 0xFFFFFFFF or 4,294,967,295

Overflow Problem

- Problem here to detect whether two unsigned 16-bit numbers would overflow when added?

```
bool IsValidAddition(unsigned short x, unsigned short y)
{
        if (x + y < x)
                return false;
        return true;
}
```

Overflow Problem in C#?

- Following code throws a compiler error, how would you fix it?

```
byte a, b;
a = 255;
b = 1;
byte c = (a + b);
```

ERROR: Cannot implicitly convert type ‘int’ to ‘byte’

Examples

- CVE-2008-0726
  - Integer overflow in Adobe Reader and Acrobat 8.1.1 and earlier allows remote attackers to execute arbitrary code via crafted arguments to the printSepsWithParams, which triggers memory corruption.
- CVE-2007-6261
  - Integer overflow in the load_threadstack function in the Mach-O loader (mach_loader.c) in the xnu kernel in Apple Mac OS X 10.4 through 10.5.1 allows local users to cause a denial of service (infinite loop) via a crafted Mach-O binary.

Spotting the Overflow Sin

- Anything doing arithmetic
- Especially if input provided by the user
- Focus especially on array index calculations

- Redemption
  - Use size_t type in C/C++
  - Use unsigned integers if appropriate, easier to verify
  - Avoid “clever” code in favor of straightforward code

Sin 4 : SQL Injection

- How do bad guys get credit card numbers from sites?
  - Break into server using exploit like buffer overrun
  - Go through open port with sysadmin password
  - Social engineering
  - SQL injection attacks

SQL Injection Example

- PHP code

```
$id = $_REQUEST[“id”];
$pass = $_REQUEST[“password”];
$qry = “SELECT ccnum FROM cust WHERE id = $id AND pass=$pass”;
```

SQL Injection Example

- PHP code

```
$id = $_REQUEST[“id”];
$pass = $_REQUEST[“password”];
$qry = “SELECT ccnum FROM cust WHERE id = ‘$id’ AND
pass=‘$pass’”;
```

User inputs id of user to attack
For password, enters: **‘ OR 1=1 –**

-- is the comment operator, to ignore whatever comes afterwards

Another:

Password:**’ OR ‘’=‘**

Spotting the Sin

- Takes user input
- Does not check user input for validity
- Uses user input data to query a database
- Uses string concatenation or string replacement to build the SQL query or uses the SQL Exec command

Examples

- CAN-2004-0348
  - SQL injection vulnerability in viewCart.asp in SpiderSales shopping cart software allows remote attackers to execute arbitrary SQL via the userID parameter
- CVE-2008-0682
  - SQL injection vulnerability in wordspew-rss.php in the Wordspew plugin for Wordpress allows remote attackers to execute arbitrary SQL commands via the id parameter.

Redemption

- Filter any unallowable characters like ‘ or “”
  - mysql_real_escape_string($var)
- Never use string concatenation to build SQL statements
  - Use stored procedures with parameters
  - C#:

```
cmd = new SqlCommand(queryName, sqlConn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add(“@ID”,Id);
```

  - PHP 5.0:

```
mysqli_stmt_bind_param($stmt, “s”, $id);
mysqli_stmt_execute($stmt);
```

    Default in PHP for GET/POST is to add slashes

Sin 5 : Command Injection

- In 1994, one could get a root shell on an SGI computer running IRIX by sending the following to a printer:

```
FRED; xterm&
```

- Code:

```
char buf[1024];
snprintf(buf, “system lpr –P %s”, user_input, sizeof(buf) -1);
system(buf);
```

Spotting the Sin

- Look for calls to system(), exec()
- Java too:
  - Class.forName(String name);
    - Dynamically load and run Java code
  - Runtime.exec()
- Redemption
  - Check the data to make sure it is ok

Sin 6 : Failing to Handle Errors

- We've already said (or will say) a fair bit about mishandling errors and how try/catch can be misused from the Code Complete book

Sin 7 : Cross Site Scripting

- Somewhat misnamed, as crossing sites is not always necessary to exploit this bug
- Sin is straightforward:
  - Web app takes input from a user
  - Input is stored or echoed back to the user
  - That's it

PHP Example

```
<?php

        if($_SERVER['REQUEST_METHOD'] != "POST")
        {
                header("Content-Type: text/html");
                print("<HTML><HEAD><TITLE>My Page</TITLE>");
                print("</HEAD>");
                print("<BODY>");
                print("<FORM method=post action='cssSin.php'>");
                print("Enter your comment.<p>");
                print("<INPUT type=text name='comment'>");
                print("<INPUT type=submit value='Submit'>");
                print("</FORM>");

                print("<HR>");
                print("<B>Here is the comment log:</B><p>");
```

PHP Example

```
                $f = fopen("c:\\comments.txt","r");
                print("<UL>");
                while (!feof($f))
                {
                        $line = fgets($f,2000);
                        print("<li>" . $line . "</LI>");
                }
                fclose($f);
                print("</UL>");
        }
        else {
                $comment = $_REQUEST['comment'];

                $f = fopen("c:\\comments.txt", "a");
                fprintf($f, "\n" . $comment);
                fclose($f);

                print("Thank you, your comment has been saved.");
        }
?>
```

CSS Problem

- Malicious user can inject script code that is then executed when another user views that page
- Even if the input is merely echoed, a malicious user might:
  - Lure victim to their page
  - Get victim to click on a link which refers victim to the vulnerable site with the CSS bug
  - Script code is run under domain of the server and could get cookies or modify any elements of the DOM like tweak all links to point to porn sites

Spotting the Sin

- The web app takes input from a header, form, or query string
- App does not check the input for validity
- App echoes back data from a user into the browser

- Redemption
  - Restrict input to valid input only
  - HTML encode the output ; < > etc.

Sin 8 : Failing to Protect Network Traffic

- Mostly skipping
- Network vulnerable to
  - Eavesdropping
  - Replay
  - Spoofing
  - Tampering
  - Hijacking
- Use SSL / TLS for session security

Sin 9 : Magic URLs and Hidden Form Fields

- Magic URLs:
  - http://www.xyz.com/?val=1&q=foo&user=n58
  - http://www.xyz.com/?id=TKSJDARJ$J14$J==
- Hidden Form Fields to pass variables

```
<form action = " …"
        <input type=text name="product">
        <input type=hidden name="price" value="300">
</form>
```

Redemption

- Use SSL or store data on server side
- Session variables, encrypted

Sin 10 : Improper Use of SSL and Transport Layer Security

- If server authentication not done properly, attacker can eavesdrop or modify conversations
  - Especially vulnerable when key associated with certificate
- Feeling that site is impenetrable simply because it uses SSL
  - Still can have overflow, SQL injection, etc…

Sin 11 : Use of Weak Passwords

- People hate passwords, it is a battle to force people to use strong passwords
- Consider password content policy
  - Length, characters, reset frequency…
  - Password storage?
    - Storage in the clear is bad; use hash+salt
  - How to recover a lost password?
    - Paris Hilton T-Mobile Sidekick phone hijack
    - Broke into server side by getting username and asking for a password reset
      - Challenge question: “What is the name of your favorite pet?”

Guidelines for Password Resets

- Locking users out of accounts for too many bad password attempts may result in DoS
- Recommendations
  - Limit number of attempts to reasonable number like 20/hour
  - Slow down authentication process after certain number of bad attempts
  - Make users provide multiple pieces of information to reset a password, might require “thing they have” like a ID card
  - Use more obscure questions

Tenex Bug

- TENEX Operating System pseudocode to validate:

```
For i = 0 to len(typed_password)
    if i >= len(actual_password)  fail;
    if typed_password[i] != actual_password[i] fail;
if i < len(actual_password) fail;
Success;
```

Flaw: Attacker could put candidate password in memory overlapping page boundaries. First letter on one page, second letter on the next, if the first letter was correct there was a pause while the page for the second letter loaded

Sin 12 : Failing to Store and Protect Data Securely

- Unix: tendency to give permissions to all
- Windows: Access Control Lists can be mind boggling as which objects to consider what can be controlled
  - Don't take the easy way out and give out too many permissions
- Don't embed secret data in code
  - E.g. passwords
  - Use DPAPI or KeyChain or at least store passwords somewhere not hard-coded in the app

Examples

- Diebold voting machine
  - Diebold's default password identification number for microchip-embedded "smartcards“ used by voting administrators was “1111”
  - California source code review found hard-coded passwords of “diebold” and “12345678”
- CAN-2004-0391
  - A default username/password pair is present in all releases of the Cisco Wireless Lan Solution Engine. A user that logs in using this username has complete control of the device. This username cannot be disabled. There is no workaround.

Sin 13 : Information Leakage

- Attacker gets information, implicitly or explicitly, that could provide more information for the attacker to reach their goal
- Examples:
  - Name of server software, versions
  - Debugging information (e.g. left on in PHP)
  - Error messages that reveal code structure

Sin 14: Improper File Access

- Watch out for race conditions among accessing files
  - Perl:

```
#!/usr/bin/perl
my $file = "$ENV{HOME}/.config";
read_config($file) if –r $file;
```

  Between the file check and read, the file may disappear if there are multiple processes handling this file
- Manipulation of pathnames to overwrite important files

Sample Bug

- CVE-2004-0115
  - Microsoft Virtual PC for Macintosh
    - The VirtualPC Services for Mac 6.0 and 6.1 allowed local attackers to truncate and overwrite arbitrary files, and potentially execute arbitrary code via a symlink attack on the /tmp/VPCServices_Log temporary file. The code blindly opens a temporary file named /tmp/VPCServices_Log regardless of whether the file is real or a symlink. If this symlink points to another file, that file is clobbered.

Redemption

- Keep all files in a place attackers can't control
- Resolve the path to the file, following symbolic links
- If opening a temp file in a public directory, add on random numbers, base64 encode it

Sin 15 : Trusting Network Name Resolution

- Not too difficult to have an unsecure name server, e.g. might use WINS
- Skipping
- Might ensure connections running over SSL

Sin 16 : Race Conditions

- Common bug in multi-threaded applications, which includes web apps
- Actual bug from a DB web application, sometimes the queries were UPDATE to a shared field

```
try
{
        sqlComm.sqlConn.Open();
        comm.ExecuteNonQuery();
        sqlComm.sqlConn.Close();
}
catch(Exception f)
{
        sqlComm.sqlConn.Close();
}
```

Sin 16 : Race Conditions

- Lock access to any query for mutual exclusion

```
lock(sqlComm.sqlConn)
{
        try
        {
                sqlComm.sqlConn.Open();
                comm.ExecuteNonQuery();
                sqlComm.sqlConn.Close();
        }
        catch(Exception f)
        {
                sqlComm.sqlConn.Close();
        }
}
```

Spotting the Sin

- Multiple threads or processes that write to the same source
- Creating files or directories in common areas
- Signal handlers

Sin 17 : Unauthenticated Key Exchange

- Skipping this one

Sin 18 : Cryptographically Strong Random Numbers

- Seeds for pseudo-random number generators may not be that difficult to regenerate, then use to test a sequence of random values and determine what the next “random” number will be
- Can try true random number generators
  - Mouse, keyboard, Random.org, etc.

Sin 19 : Poor Usability

- We will cover plenty about this one ☺
- Poor usability can also mean poor security
  - Always clicking “OK” when given lots of dialogs
  - Cryptic error or status messages

CWE/Sans Top 25 Most Dangerous Programming Errors

Jan 2009

Top 25 Errors

- The Common Weakness Enumeration (CWE) is a formal list of software weakness types and is sponsored by the US Department of Homeland Security's National Cyber Security Division
- The SANS (SysAdmin, Audit, Network, Security) Institute was established in 1989 as a cooperative research and education organization
- Source: http://www.sans.org/top25errors/

Contributors

- Robert C. Seacord, CERT
- Pascal Meunier, CERIAS, Purdue University
- Matt Bishop, University of California, Davis
- Kenneth van Wyk, KRvW Associates
- Masato Terada, Information-Technology Promotion Agency (IPA), (Japan)
- Sean Barnum, Cigital, Inc.
- Mahesh Saptarshi and Cassio Goldschmidt, Symantec Corporation
- Adam Hahn, MITRE
- Jeff Williams, Aspect Security
- Carsten Eiram, Secunia
- Josh Drake, iDefense Labs at VeriSign, Inc.
- Chuck Willis, MANDIANT
- Michael Howard, Microsoft
- Bruce Lowenthal, Oracle Corporation
- Mark J. Cox, Red Hat Inc.
- Jacob West, Fortify Software
- Djenana Campara, Hatha Systems
- James Walden, Northern Kentucky University
- Frank Kim, ThinkSec
- Chris Eng and Chris Wysopal, Veracode, Inc.
- Ryan Barnett, Breach Security
- Antonio Fontes, New Access SA, (Switzerland)
- Mark Fioravanti II, Missing Link Security Inc.
- Ketan Vyas, Tata Consultancy Services (TCS)
- Lindsey Cheng, Ian Peters and Tom Burgess, Secured Sciences Group, LLC
- Hardik Parekh and Matthew Coles, RSA - Security Division of EMC Corporation Mouse
- Ivan Ristic Apple Product Security
- Software Assurance Forum for Excellence in Code (SAFECode)
- Core Security Technologies Inc.
- Depository Trust & Clearing Corporation (DTCC)

Kudos

- National Security Agency's Information Assurance Directorate
  - "The publication of a list of programming errors that enable cyber espionage and cyber crime is an important first step in managing the vulnerability of our networks and technology. There needs to be a move away from reacting to thousands of individual vulnerabilities, and to focus instead on a relatively small number of software flaws that allow vulnerabilities to occur, each with a general root cause. Such a list allows the targeting of improvements in software development practices, tools, and requirements to manage these problems earlier in the life cycle, where they can be solved on a large scale and cost-effectively.“
  - Tony Sager, National Security Agency's Information Assurance Directorate

Kudos

- Microsoft:
  - "The 2009 CWE/SANS Top 25 Programming Errors project is a great resource to help software developers identify which security vulnerabilities are the most important to understand, prevent and fix.“
  - Michael Howard, Principal Security Program Manager, Security Development Lifecycle Team, Microsoft Corp.
- Symantec:
  - "The 2009 CWE/SANS Top 25 Programming Errors reflects the kinds of issues we've seen in application software and helps provide us with actionable direction to continuously improve the security of our software."
  - - Wesley H. Higaki, Director, Software Assurance, Office of the CTO, Symantec Corporation

Insecure Interaction Among Components

- CWE-20: Improper Input Validation
  - It's the number one killer of healthy software, so you're just asking for trouble if you don't ensure that your input conforms to expectations…
- CWE-116: Improper Encoding or Escaping of Output
  - Computers have a strange habit of doing what you say, not what you mean. Insufficient output encoding is the often-ignored sibling to poor input validation, but it is at the root of most injection-based attacks, which are all the rage these days...
- CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')
  - If attackers can influence the SQL that you use to communicate with your database, then they can...

Insecure Interaction Among Components

- CWE-79: Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
  - Cross-site scripting (XSS) is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications...If you're not careful, attackers can...
- CWE-78: Failure to Preserve OS Command Structure (aka 'OS Command Injection')
  - When you invoke another program on the operating system, but you allow untrusted inputs to be fed into the command string that you generate for executing the program, then you are inviting attackers...
- CWE-319: Cleartext Transmission of Sensitive Information
  - If your software sends sensitive information across a network, such as private data or authentication credentials, that information crosses many...

Insecure Interaction Among Components

- CWE-352: Cross-Site Request Forgery (CSRF)
  - With cross-site request forgery, the attacker gets the victim to activate a request that goes to your site. Thanks to scripting and the way the web works in general, the user might not even be aware that the request is being sent. But once the request gets to your server, it looks as if it came from the user, not the attacker.
- CWE-362: Race Condition
  - Attackers will consciously look to exploit race conditions to cause chaos or get your application to cough up something valuable...
- CWE-209: Error Message Information Leak
  - If you use chatty error messages, then they could disclose secrets to any attacker who dares to misuse your software. The secrets could cover a wide range of valuable data...

Risky Resource Management

- CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer
  - Buffer overflows are Mother Nature's little reminder of that law of physics that says if you try to put more stuff into a container than it can hold, you're...
- CWE-642: External Control of Critical State Data
  - There are many ways to store user state data without the overhead of a database. Unfortunately, if you store that data in a place where an attacker can access it...
- CWE-73: External Control of File Name or Path
  - When you use an outsider's input while constructing a filename, you're taking a chance. If you're not careful, an attacker could...

Risky Resource Management

- CWE-426: Untrusted Search Path
  - If a resource search path (e.g. path to JAR file) is under attacker control, then the attacker can modify it to point to resources of the attacker's choosing. This causes the software to access the wrong resources at the wrong time...
- CWE-94: Failure to Control Generation of Code (aka 'Code Injection')
  - For ease of development, sometimes you can't beat using a couple lines of code to employ lots of functionality. It's even cooler when the code is executed dynamically…
- CWE-494: Download of Code Without Integrity Check
  - You don't need to be a guru to realize that if you download code and execute it, you're trusting that the source of that code isn't malicious. But attackers can perform all sorts of tricks...

Risky Resource Management

- CWE-404: Improper Resource Shutdown or Release
  - When your precious system resources (e.g. allocated memory) have reached their end-of-life, you need to dispose of them correctly…
- CWE-665: Improper Initialization
  - Just as you should start your day with a healthy breakfast, proper initialization helps to ensure your attacker doesn't initialize your data for you…
- CWE-682: Incorrect Calculation
  - When attackers have some control over the inputs that are used in numeric calculations, this weakness can lead to vulnerabilities. It could cause you to make incorrect security decisions. It might cause you to...

Porous Defenses

- CWE-285: Improper Access Control (Authorization)
  - If you don't ensure that your software's users are only doing what they're allowed to, then attackers will try to exploit your improper authorization and...
- CWE-327: Use of a Broken or Risky Cryptographic Algorithm
  - You may be tempted to develop your own encryption scheme in the hopes of making it difficult for attackers to crack. This kind of grow-your-own cryptography is a welcome sight to attackers...
- CWE-259: Hard-Coded Password
  - Hard-coding a secret account and password into your software's authentication module is...

Porous Defenses

- CWE-732: Insecure Permission Assignment for Critical Resource
  - If you have critical programs, data stores, or configuration files with permissions that make your resources accessible to the world - well, that's just what they'll become...
- CWE-330: Use of Insufficiently Random Values
  - If you use security features that require good randomness, but you don't provide it, then you'll have attackers laughing all the way to the bank...

Porous Defenses

- CWE-250: Execution with Unnecessary Privileges
  - Spider Man, the well-known comic superhero, lives by the motto "With great power comes great responsibility." Your software may need special privileges to perform certain operations, but wielding those privileges longer than necessary can be extremely risky...
- CWE-602: Client-Side Enforcement of Server-Side Security
  - Remember that underneath that fancy GUI, it's just code. Attackers can reverse engineer your client and write their own custom clients that leave out certain inconvenient features like all those pesky security controls...