
Special Topics in Security

ECE 5698

Engin Kirda
ek@ccs.neu.edu



Northeastern University

Multi User OSs (Windows XP, NT, 2003)

- Obviously – notion of multiple users, multiple tasks
- Authentication
- Access Control
- Privilege Management
- Accounting, Quotas
- Windows NT, XP, 2003
 - object-oriented
 - Systems are based on similar technologies and code-base (i.e., vulnerabilities are usually across multiple platforms)
 - Security Monitor, tightly coupled host and network security





Windows NT Passwords

- NT Maintains backward compatibility to Win95/98, NT passwords can be easy to crack
 - A LANMAN password hash is upper cased, padded to 14 characters, divided into two seven character parts, each of which is used as a key to encrypt a constant.
 - After LANMAN passwd hash is cracked, 2 to the nth power (where n is the length of the password) gives the maximum number of case variations that must be tried to get the NT password (about a second ;-)).
 - LANMAN authentication could only be partially disabled (i.e., logging in). The passwd storage scheme and encryption was still weak against brute force attacks.
- In 2001, method was provided to disable LANMAN hashes on 2000 and XP, but not NT



NT and 98 Threat Mitigation Guide

- Syn flooding protection registry hacks
 - Syn flooding is a common attack. Each connection request requires server to allocate certain amount of memory and kernel structures
 - HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\SynAttackProtect(RegDword) = 1
 - HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpMaxHalfOpen(RegDword) = 100 (maximum number of connections)
- Tips and tricks to configure your system (e.g., ICMP redirects, router discovery, RPC stuff)



NT and 98 Threat Mitigation Guide

- NSA has developed a set of recommended permissions for NT servers and workstations
 - “Guide to securing MS Windows NT networks”
- Restrict null-session and anonymous accesses (i.e., to the registry)
- Prevention of the storage of LANMAN passwords
- Patching is recommended by MS, but sometimes was not possible (i.e., on illegal copies... was this a good idea?)



Windows XP Service Pack 2

- A set of security “technologies” for improvement of situation
 - Network protection
 - Improved firewall, reduced RPC attack surface (reduced credentials)
 - Memory protection
 - MS version of StackGuard has been deployed (/GS switch)
 - E-mail handling
 - Outlook has been “fixed”, recompiled
 - Web browsing security
 - Privileges / locking down
 - Automatic updates (there were reports of problems at first)

Was Windows XP SP2 more secure?

- “It is late, it is large, but a step in the right direction”
However...
 - Malware writers have not given up, and continue to try to compromise and evade
- Enjoy XP SP2 firewall with care: Only *inbound* connections are checked
 - It is possible for code to modify setting and firewall (e. Phrack articles). See recent postings.



Windows Vista Security

- The “wow!” effect ;-)
 - Vista == You take Mac OS UI gimmicks and combine them with new features ;)
- Now, most applications run in non-admin mode
 - User account control: If privilege is required, Secure Desktop mode is activated
- Bitlocker Drive encryption, Encryption FS
 - Full volume encryption, key can be stored on USB



Windows Vista Security

- Windows Firewall
 - Was improved (e.g., outbound connections, port ranges, IP ranges, etc.)
- Windows Defender
 - Microsoft's anti-spyware utility was included
- Windows Parental Controls
 - Web content blocking, time limitations on account, restrictions on programs executed, etc.
- Exploit prevention
 - Address Space Layout Randomization (ASLR)
 - Encryption of function pointers
 - Stack overflow detection (canary mechanism)



Windows Vista™



Windows Vista Security

- Data Execution Prevention (DEP)
 - Vista supports full support for NX feature of processor
 - Problem: Not all applications / processors are DEP-aware
- Application isolation
 - Mandatory Integrity Control
 - Application in a lower integrity level cannot access resources in higher integrity level
- Network Access Protection (NAP)
 - Computers should conform to preset “system health” level, otherwise, network access limited or denied (e.g., updates need to be installed)

Windows Vista Security

- Process Isolation
 - Previous versions of Windows, all services ran under same session
 - Not so anymore: Isolation Session 0
 - Normal process cannot show popups or dialogs anymore
 - If it does, it will be invisible and will sit in the background
 - To interact, processes need to use Windows calls (so there is stricter control)



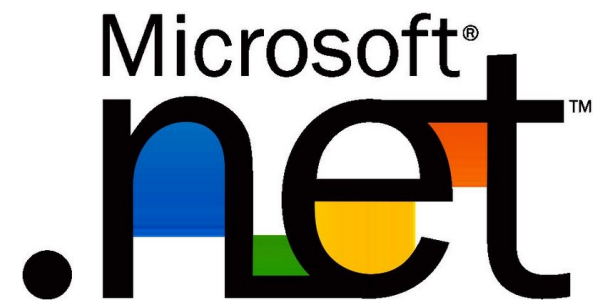


Windows Vista Security

- File and registry virtualization
 - Windows programmers generally assumed that they are admin
 - Thousands of programs exist out there so backwards compatibility is important
 - However, all-access registry operations have been disabled
 - Microsoft has introduced a file and registry virtualization for backwards-compatibility
 - Application writes to a “per user” location, does not realize it, it is transparent

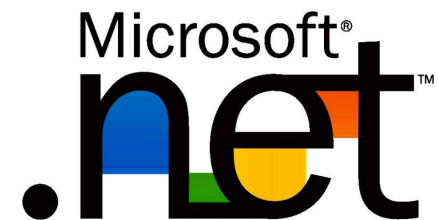
.NET Framework Security

- Managed execution and type safety
 - Exception manager
 - Buffer overflows not possible
 - Security Engine
 - Code Access Security
 - But wait... there is “unmanaged” mode...
- CLR Integrated Security
 - Code access security
 - Role-based security
- .NET Framework Libraries
 - Cryptography
 - Web Services and Applications



.NET Framework Security

- Remote code:
 - With the growth of the Internet, applications are increasingly downloaded from remote sources
 - Users are susceptible to executing malicious code
- The proposed solution by Microsoft:
 - Introduced the .NET framework, where machine-independent byte-code is executed on a virtual machine
 - .NET is an implementation of the Common Language Infrastructure (CLI)
 - Consists of Common Type System (CTS)
 - .NET is type-safe and memory-safe



.NET Framework Security

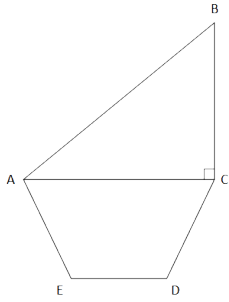
- An important feature of .NET
 - It allows access to native libraries (i.e., legacy code support)
 - .NET applications are called *managed* and native code is referred to as *unmanaged* code
 - The runtime environment can enforce security restrictions by relying on type and memory-safety
 - Security model is called Code Access Security (CAS)
 - CAS uses *evidence* provided by the program and security policies to generate permissions (e.g., file access)
- Unfortunately, the execution of unmanaged native code is not restricted by the security model
 - Hence, an attacker can completely circumvent the .NET security mechanisms

Invoking Unmanaged Code in .NET

- To support interoperability with languages such as C, C++
 - CLI uses a mechanism called *Platform Invoke Service* (P/Invoke)
 - Because native code can modify the security state of user's environment, .NET permissions are *full trust*
 - The native code is run within the same process as CLI, an attacker could modify .NET runtime itself
 - Microsoft suggests P/Invoke to be used for highly-trusted code
 - This, however, cannot always be feasible

Web Security I





Web Application Security

- When an organization puts up a web application, they invite everyone to send them HTTP requests.
- Attacks buried in these requests sail past firewalls without notice because they are inside legal HTTP requests.
- Even “secure” websites that use SSL just accept the requests that arrive through the encrypted tunnel without scrutiny.
- **This means that your web application code is part of your security perimeter!**

Web Application Security

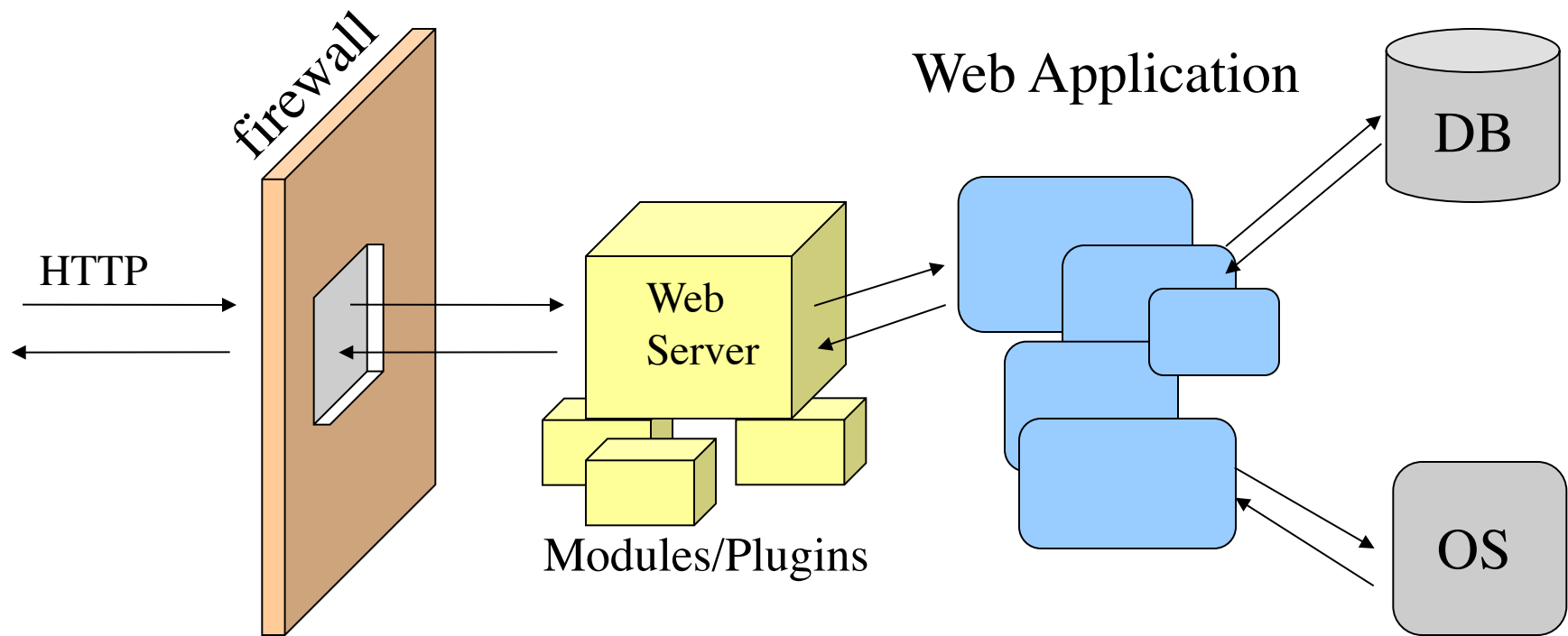
- The security issues related to the Web are not new. In fact, some have been well understood for decades
 - For a variety of reasons, major software development projects are still making these mistakes and jeopardizing not only their customers' security, but also the security of the entire Internet.
 - There is no “silver bullet” to cure these problems. Today's assessment and protection technology is improving, but can currently only deal with a limited subset of the issues at best.
 - To address the security issues, organizations will need to change their development culture, train developers, update their software development processes, and use technology where appropriate.

Why It is Important

- Easiest way to compromise hosts, networks and users
- Widely deployed
- No Logs! (POST Request payload)
- Difficult to defend against or to detect
- Firewall? What firewall? I don't see any firewall...
- Encrypted transport layer does not help much



On a Typical Web Server



On a typical Web server...

- Your host has an open 80/8080 port (firewall)
- Following components are running
 - OS
 - Web Server
 - Main application (e.g., Apache)
 - Plugins
 - Servlets
 - Scripts (CGI, Perl, ...)

HTTP and Web Application Basics

- All HTTP transactions follow the same general format. Each client request and server response has three parts: the request or response line, a header section, and the entity body. The client initiates a transaction as follows:
 - *GET /index.html?param=value HTTP/1.0*
- After sending the request and headers, the client may send additional data. This data is mostly used by CGI programs using the POST method.
 - Note that for the GET method, the parameters are encoded into the URL



HyperText Transfer Protocol (HTTP)

```
> nc www.iseclab.org 80
GET /index.html HTTP/1.1
HOST: www.iseclab.org

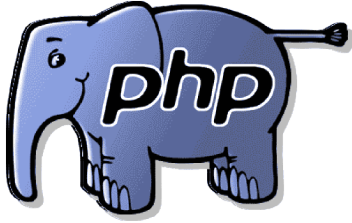
HTTP/1.1 200 OK
Date: Sun, 29 Mar 2009 09:28:06 GMT
Server: Apache/2.2.8 (Ubuntu) mod_python/3.3.1 Python/2.5.2 PHP/5.2.4-2ubuntu5.5 with
Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g
Last-Modified: Wed, 25 Mar 2009 19:20:22 GMT
ETag: "4c072-4d82-465f664106980"
Accept-Ranges: bytes
Content-Length: 19842
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```




HTTPS

- Combination of HTTP and SSL (or TLS)
- Encrypt the communication
 - Protect against eavesdropping
 - Protect from man in the middle attacks (provided that the certificate is trusted)
 - Used to protect the user authentication
 - By-pass IDS / IPS
- Does not protect from attacks against the web application



Web Server Scripting

- Allows easy implementation of functionality (also for non-programmers – Think: Is this good?)
- Example scripting languages are Perl (e.g., used in the InetSec challenges), Python, ASP, JSP, PHP
- Scripts are installed on the Web server and return HTML as output that is then sent to the client
- Template engines are often used to power Web sites
 - E.g., Cold Fusion, Cocoon, Zope
 - These engines often support/use scripting languages

just
another
example

Web Application Example

- Objective: To write an application that accepts a username and password and prints (displays) them
 - First, we write HTML code and use forms

```
<html><body>  
  <form action="/scripts/login.pl" method="post">  
    Username: <input type="text" name="username"> <br>  
    Password: <input type="password" name="password"> <br>  
    <input type="submit" value="Login" name="login">  
  </form>  
</body></html>
```

Web Application Example

- Second, here is the corresponding Perl script that prints the username and password passed to it:

```
#!/usr/local/bin/perl
uses CGI;
$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
...
print "<html><body> Username: $username <br>
Password: $password <br>
</body></html>";
```

A Common Root for Many Problems

- Web applications use input from HTTP requests (and occasionally files) to determine how to respond
 - Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms
 - Common input tampering attempts include XSS, SQL Injection, hidden field manipulation, buffer overflows, cookie poisoning, hidden field manipulation, remote file inclusion...
- Some sites attempt to protect themselves by filtering known malicious input
 - Problem:
there are many different ways of encoding information

Unvalidated Input

- A surprising number of web applications use only client-side mechanisms to validate input
 - Client side validation mechanisms are easily bypassed, leaving the web application without any protection against malicious parameters
- How to determine if you are vulnerable?
 - Any part of an HTTP request that is used by a web application without being carefully validated is known as a “tainted” parameter
 - The simplest way: to have a detailed code review, searching for all the calls where information is extracted from an HTTP request

Unvalidated Input

- How to protect yourself?
 - The best way to prevent parameter tampering is to ensure that all parameters are validated before they are used.
 - A centralized component or library is likely to be the most effective, as the code performing the checking should be all in one place.
- Parameters should be validated against a “positive” specification that defines:
 - Data type (string, integer, real, etc...); Allowed character set; Minimum and maximum length; Whether null is allowed; Whether the parameter is required or not; Whether duplicates are allowed; Numeric range; Specific legal values (enumeration); Specific patterns (regular expressions)

Injection Attacks: Overview

- Many webapp invoke interpreters
 - SQL
 - Shell command
 - Sendmail
 - LDAP
 - ...
- Interpreters execute the commands specified by the parameters or input data
 - If the parameters are under control of the user and are not properly sanitized, the user can inject its own commands in the interpreter

Discovering “clues” in HTML code

- Developers are notorious for leaving statements like FIXME's, Code Broken, Hack, etc... inside the source code. Always review the source code for any comments denoting passwords, backdoors, or that something doesn't work right.
- Hidden fields (`<input type="hidden"...>`) are sometimes used to store temporary values in Web pages. These can be changed with ease (Hidden Field Tampering!)

SQL Injections

SQL injection is a particularly widespread and dangerous form of injection attack that consists in **injecting SQL commands into the database engine** through an existing application

Relational Databases

- A relational database contains one or more tables
 - Each table is identified by a name
 - Each table has a certain number of named columns
- Tables contain records (rows) with data
- For example, the following table (called "users") contains data distributed in three rows

userID	Name	LastName	Login	Password
1	John	Smith	jsmith	hello
2	Adam	Taylor	adamt	qwerty
3	Daniel	Thompson	dthompson	dthompson

SQL

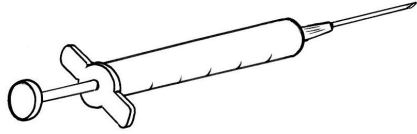
- SQL (Structured Query Language) is a language to access databases
- SQL can:
 - Queries the content of a database
 - Retrieve data from a database
 - Insert/Delete/Update records in a database
- SQL is standard (ANSI and ISO) but most DBMS implement the language in a different way, providing their own proprietary extensions in addition to the standard



SQL

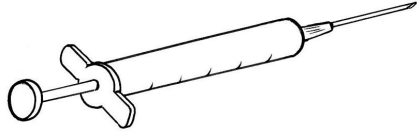
- To extract the last name of a user from the previous table:

```
mysql> SELECT LastName FROM users WHERE UserID = 1;
+-----+
| LastName |
+-----+
| Smith    |
+-----+
1 row in set (0.00 sec)
```



SQL Injections

- To exploit a SQL injection flaw, the attacker must find a parameter that the web application uses to construct a database query
- By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database
- The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents



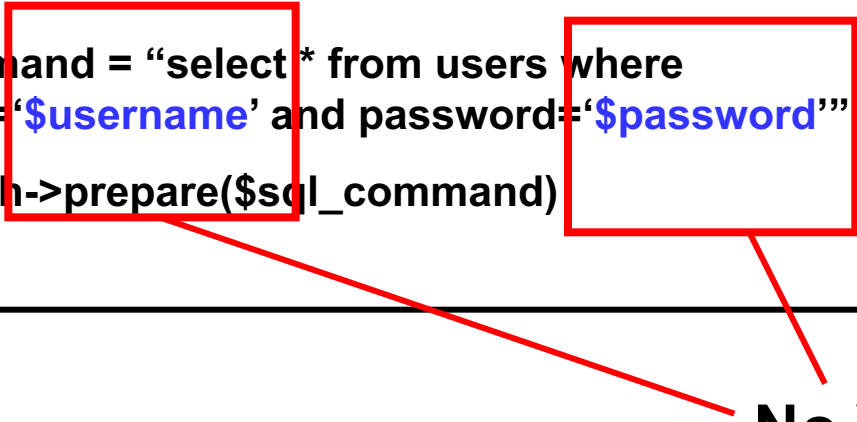
SQL Injections

- It is not a DB or web server problem
It is a flaw in the web application!
 - Many programmers are still not aware of this problem
 - Many of the tutorials and demo “templates” are vulnerable
 - Even worse, many of solutions posted on the Internet are not good enough

Simple SQL Injection Example

- Perl script that looks up *username* and *password*:

```
...  
$query = new CGI;  
$username = $query->param("username");  
$password = $query->param("password");  
...  
$sql_command = "select * from users where  
username='username' and password='password'"  
$sth = $dbh->prepare($sql_command)  
...
```



No Validation!

Simple SQL Injection Example

- If the user enters a ' (single quote) as the password, the SQL statement in the script would become:
 - select * from users where username=' ' and password = ''
 - An SQL error message would be generated
- If the user enters (injects): ' or username='john' as the password, the SQL statement in the script would become:
 - select * from users where username=' ' and password = '' or username= 'john'
 - Hence, a *different* SQL statement has been injected than what was originally intended by the programmer!

Obtaining Information using Errors

- Errors returned from the application might help the attacker (e.g., ASP – default behavior)
 - Username: ' union select sum(id) from users--
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
/process_login.asp, line 35
- Make sure that you do not display unnecessary debugging and error messages to users.
 - For debugging, it is always better to use log files (e.g., error log).

Not good!

Some SQL Attack Examples

- `select * ...; insert into user values("user","h4x0r");`
 - Attacker inserts a new user into the database
- The attacker could use “stored procedures” (e.g., in SQL Server)
 - `xp_cmdshell()`
 - “bulk insert” statement to read any file on the server
 - e-mail data to the attacker’s mail account
 - Play around with the registry settings
- `select * ... ; drop table SensitiveData;`
- Appending “;” character does not work for all databases. Might depend on the driver (e.g., MySQL)

Advanced SQL Injection

- Web applications will often escape the ' and " characters (e.g., PHP).
 - This will prevent most SQL injection attacks... but there might still be vulnerabilities
- In large applications, some database fields are not strings but numbers. Hence, ' or " characters not necessary (e.g., ... where id=1)
- Attacker might still inject strings into a database by using the "char" function (e.g., SQL Server):
 - insert into users values(666,char(0x63)+char(0x65)...)

Exploit of a Mom (xkcd)

