Essentials of

HACK-RESISTANT APPS

Martin Parry
Developer Evangelist
Microsoft
Martin.Parry@microsoft.com

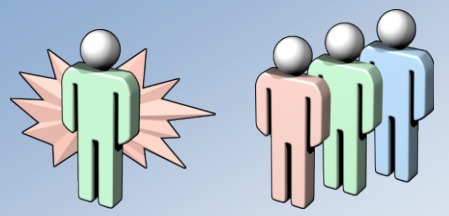
Agenda

- Justification
- Common attack methods
- Fighting back

Isn't Security for Admins?

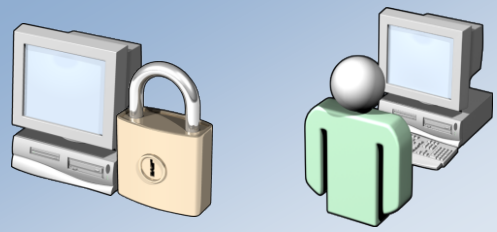
- System administrators do a lot
  - Password policy
  - Firewalls
  - Application deployment
- Things they can't do: -
  - Ensure that allowed traffic is all "good"
  - Test applications for vulnerability
  - Control the level of privilege that apps require

Challenges



Attackers vs. Defenders

- Attacker needs to understand only one security issue
- Defender needs to secure all entry points
- Attacker has unlimited time
- Defender works with time and cost constraints



Security vs. Usability

- Secure systems are more difficult to use
- Complex and strong passwords are difficult to remember
- Users prefer simple passwords

Security As an Afterthought

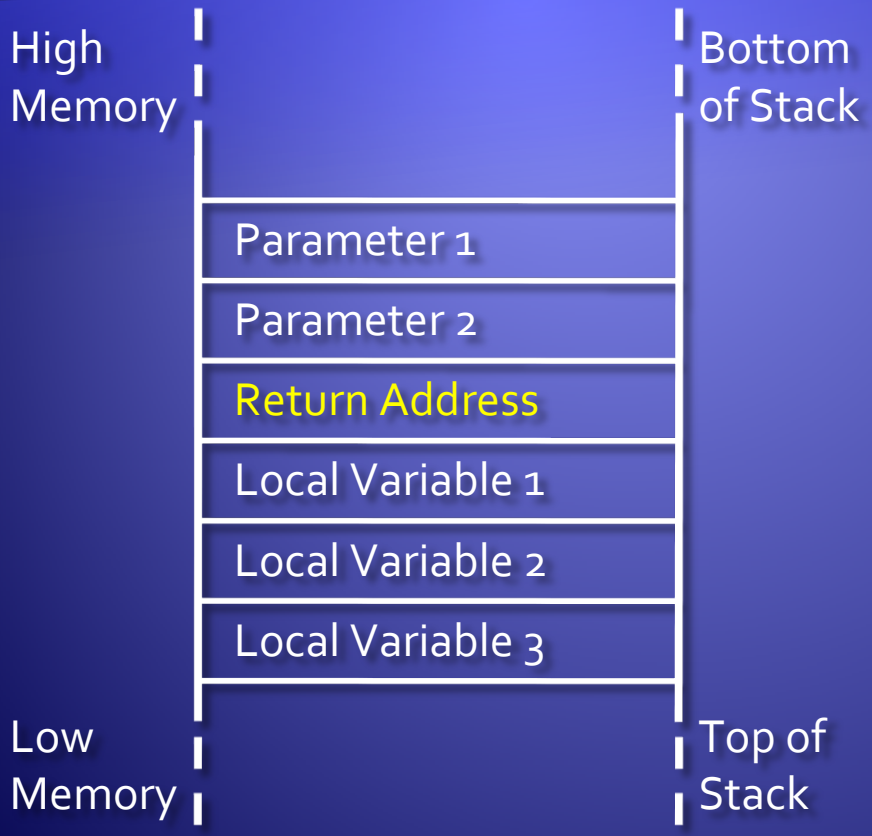
- Developers and management think that security does not add any business value
- Addressing security issues just before a product is released is very expensive

Common Vulnerabilities

The Buffer Overrun

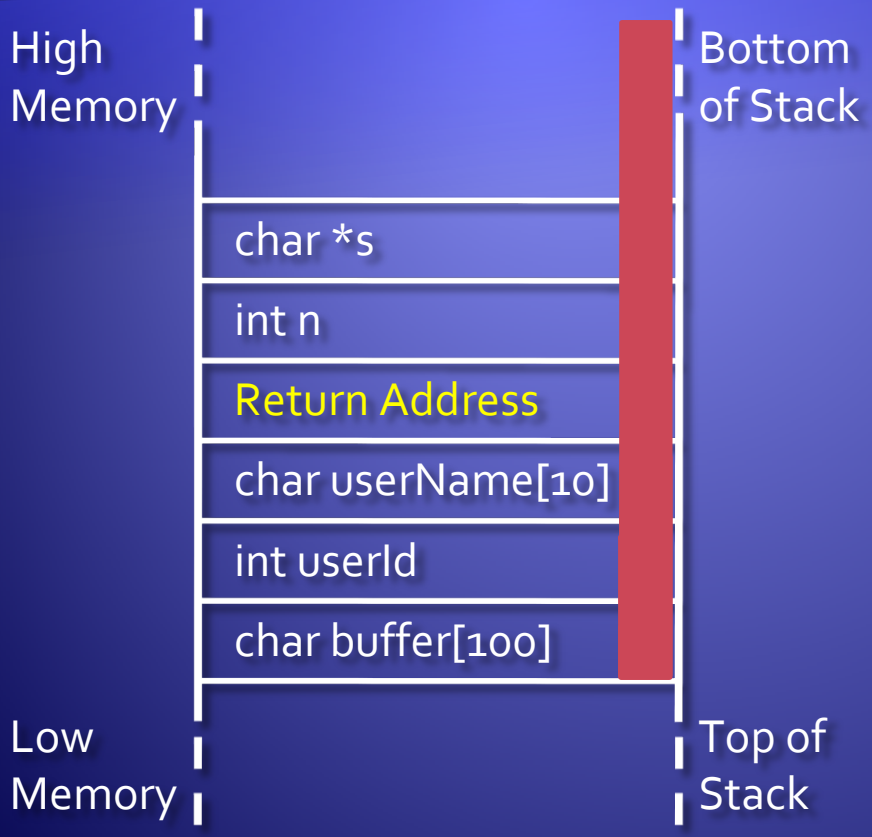
- Typically means copying input into a fixed-size buffer, without checking the input size
- Buffers can be anywhere in memory, although exploitability may differ
- There are a number of effective strategies to avoid these attacks…
- …but you have to remember to use them

The Buffer Overrun



- The Stack Frame
- What if one of the locals is a buffer?
- Exploits include
  - DoS
  - Modified behaviour

The Buffer Overrun



- Parameter s points to some input data
- Function copies that data into buffer

The Buffer Overrun

- Most vulnerable language is C++
  - Also seen in VB6
  - Any language that permits copying data in memory
- Vulnerabilities occur through rushed code
  - Friday-night, developer in a hurry, it happens!

Countering Buffer Overruns

- Use the .NET application platform
  - JIT compilation checks for buffer overruns
  - Avoid "unsafe" code in C#
- C++, use the /GS switch
  - Spots stack misuse at runtime
- Windows DEP (data execution prevention)
  - Makes use of hardware/software page marking
  - Memory pages can be marked as "data", meaning no code can be executed in them
- Code/Security Reviews

SQL Injection

- Use of unchecked input in dynamic SQL
- An unsophisticated attack with potentially devastating consequences
- All languages, all databases are vulnerable
- There are good techniques for avoiding SQL injection

Demo

SQL Injection

Countering SQL Injection

- Don't copy input straight into SQL statements
- Parameterize all commands
  - Parameter values are not compiled
- Better yet, use stored procedures
  - With parameters
  - Can deny access to underlying tables
- Code/Security Reviews

Cross-site Scripting

- Web page input reflected directly into output
  - Query string or form parameters
- Particularly dangerous when output into an HTML element the user might click on
  - Can feed in "onclick" script
  - Might forward form parameters to another site
  - Might forward cookie contents to another site
- Any web server that supports dynamic content is susceptible

Countering Cross-site Scripting

- HTMLEncode or URLEncode data before output
  - Any “special” characters are escaped
  - Still leaves some potential exploits
- Code/Security review
- Don’t echo input to output at all

Elevated Privilege

- If you're out to compromise a component, you'll look for one with high privilege
- No specific attack mechanism, but look out for this where a host process runs application code
  - IIS in-process applications
- Impersonation can be fragile
- Any compromise is potentially more serious if it affects highly-privileged code

Countering Elevated Privilege

- Well known doctrine: -
  - Run with just enough privilege to get the job done, and no more
- IIS5 – use medium or high isolation
- Windows Server 2003 – no user code runs as SYSTEM by default
- Use LocalService and NetworkService accounts for low-privileged "service" processes
- Code/Security review

Spot the Similarity

Always check your input!

Perform Code/Security Reviews

General Principles

- Expect all input to have come from a bad guy
- Expect all output to be going to a bad guy
  - Protect your secrets
- Use the Principle of Least Privilege
- Don't "roll your own" security
  - Use tried-and-tested, industry-recognized standards
- Consider moving to .NET
  - Verification, Code-access security

Minimize Attack Surface

- Expose only limited, well-documented interfaces from your application
- Use only services that your app really needs
  - Slammer and CodeRed would not have happened if certain services were off by default
  - ILoveYou (and others) would not have happened if scripting was disabled by default
- Turn off everything else

Fail Intelligently

```
DWORD dwRet = IsAccessAllowed(…);
if (dwRet == ERROR_ACCESS_DENIED) {
   // Security check failed.
      // Inform user that access is denied
} else {
   // Security check OK.
      // Perform task…
}
```

What if IsAccessAllowed() returns ERROR_NOT_ENOUGH_MEMORY?

- If your code does fail, make sure it fails securely

Fail Intelligently

- Do not:
  - Reveal information in error messages

```
<customErrors mode="On"/>
```

  - Consume resources for lengthy periods after a failure
- Do:
  - Use exception-handling blocks to avoid propagating errors back to the caller
  - Write suspicious failures to an event log

Process Techniques

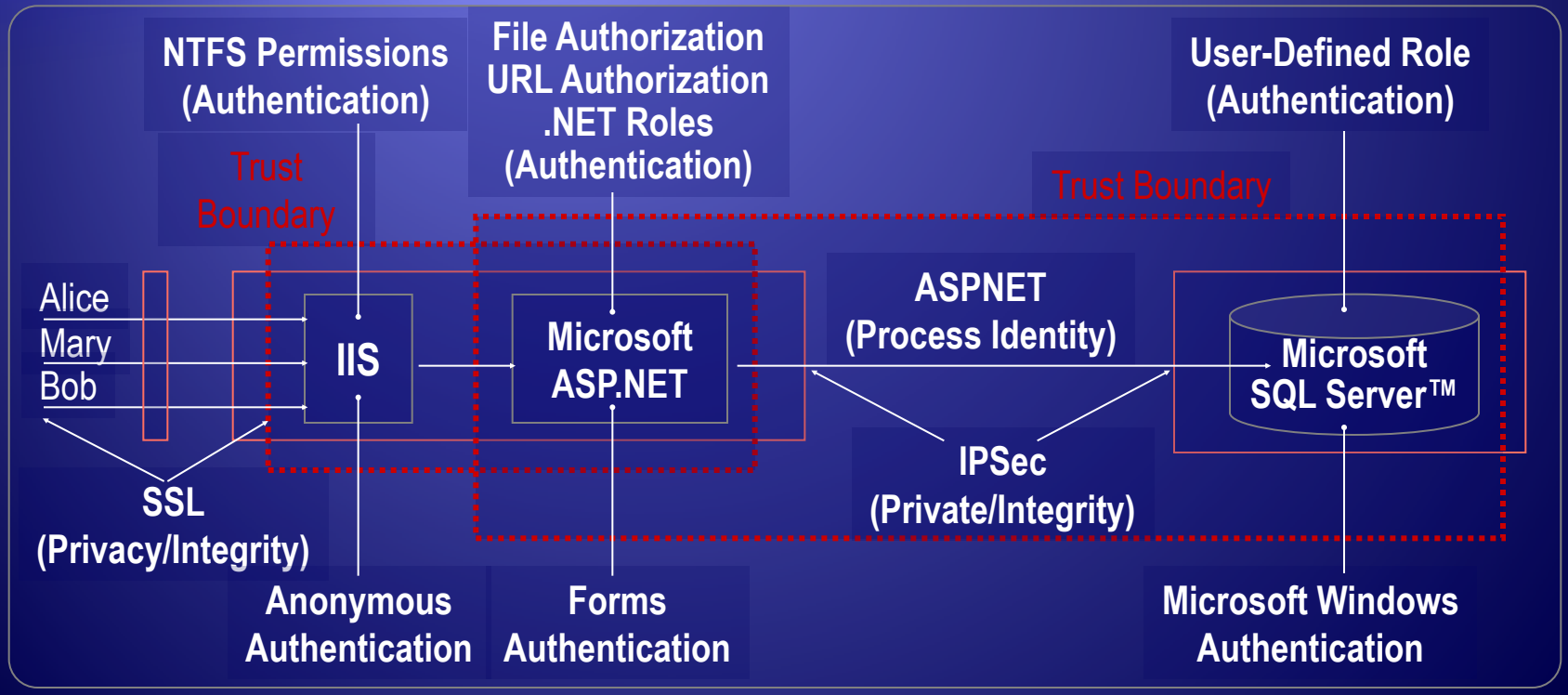
- Risk Analysis
  - Evaluates risk of compromise throughout project
- Threat Modelling
  - Helps to enumerate and prioritise threats
- The Security Development Lifecycle

Threat Modelling Process
Step 1: Identify Assets

- Build a list of assets that require protection, including:
  - Confidential data, such as customer databases
  - Web pages
  - System availability
  - Anything else that, if compromised, would prevent correct operation of your application

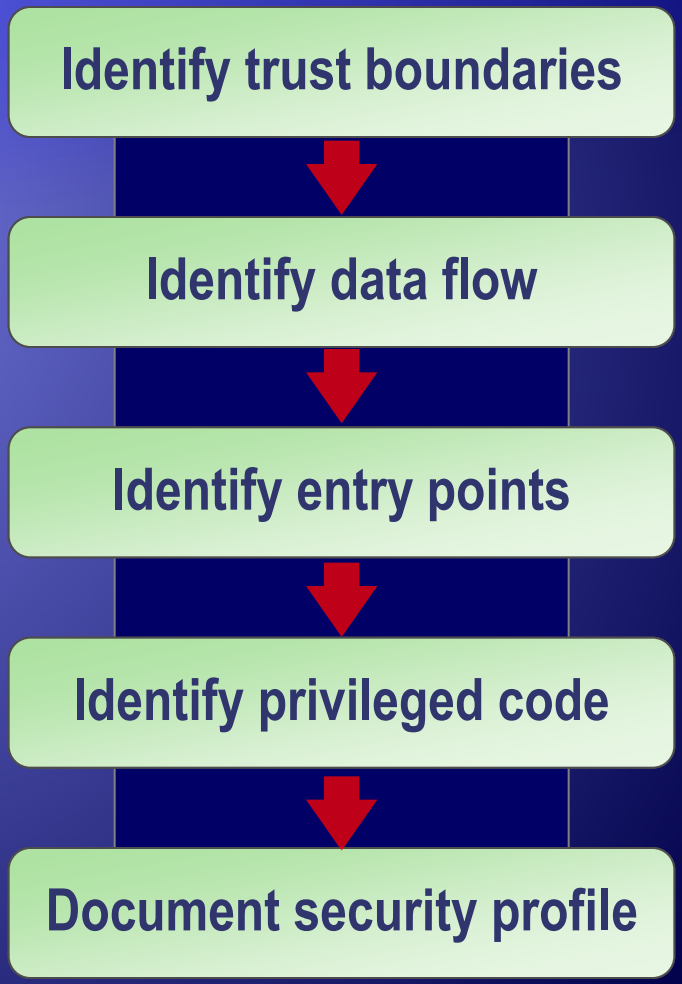
Threat Modelling Process
Step 2: Create Architecture Overview

- Identify what the application does
- Create an application architecture diagram
- Identify the technologies



Threat Modelling Process
Step 3: Decompose the Application

- Break down the application
- Create a security profile based on traditional areas with security issues
- Examine interactions between different subsystems
- Use DFD or UML diagrams



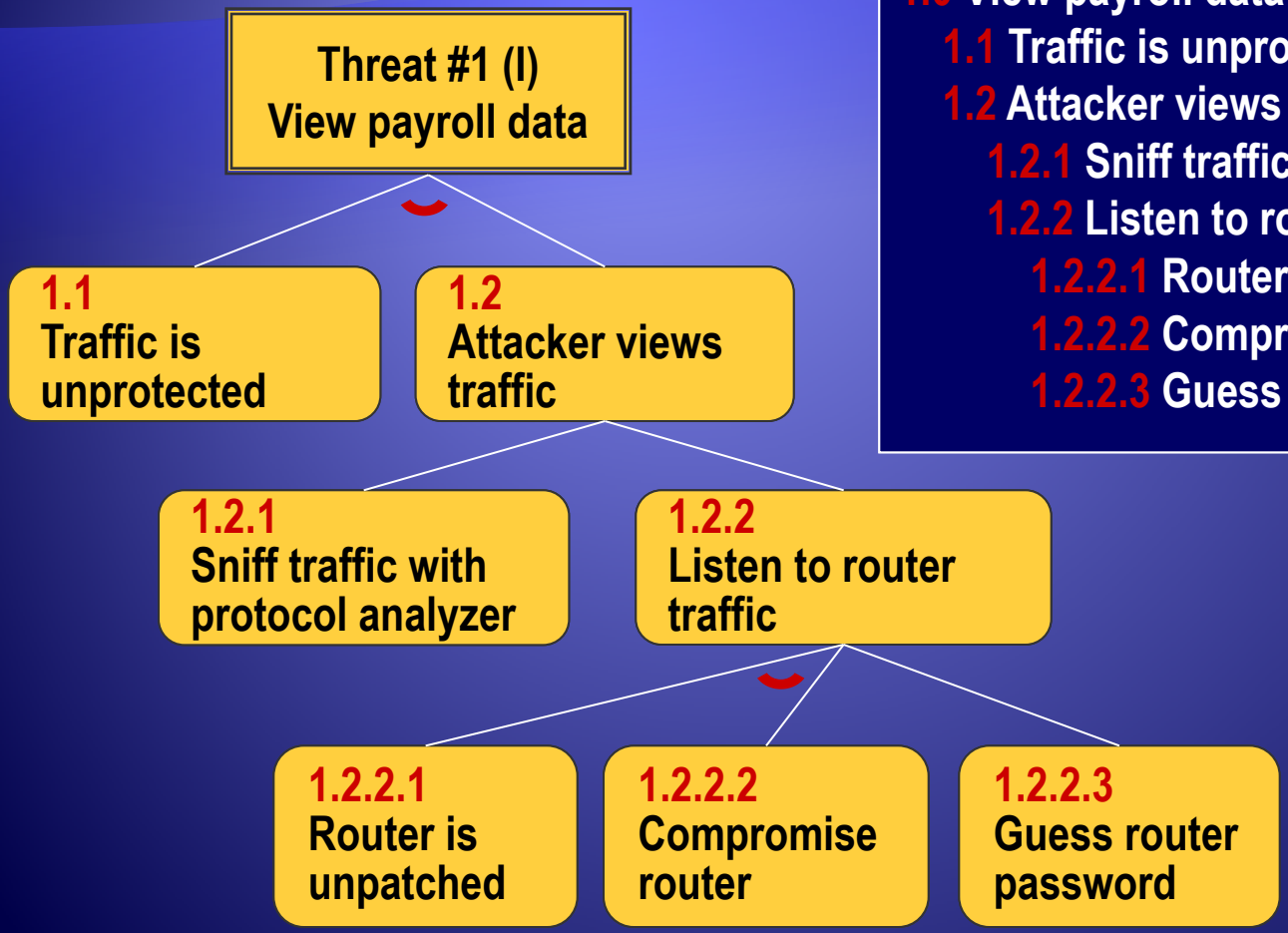
Threat Modelling Process
Step 4: Identify the Threats

- Assemble team
  - Identify roles
  - Who judges risk?
  - Who decides what an asset is?
- Identify threats
  - Network threats
  - Host threats
  - Application threats

Threat Modelling Process
Identify the Threats by Using STRIDE

Types of threats	Examples
Spoofing	• Forging e-mail messages • Replaying authentication packets
Tampering	• Altering data during transmission • Changing data in files
Repudiation	• Deleting a critical file and denying it • Purchasing a product and denying it
Information disclosure	• Exposing information in error messages • Exposing code on Web sites
Denial of service	• Flooding a network with SYN packets • Flooding a network with forged ICMP packets
Elevation of privilege	• Exploiting buffer overruns to gain system privileges • Obtaining administrator privileges illegitimately

Threat Modelling Process
Identify the Threats Using Attack Trees



1.0 View payroll data (I)
- 1.1 Traffic is unprotected (AND)
- 1.2 Attacker views traffic
  - 1.2.1 Sniff traffic with protocol analyzer
  - 1.2.2 Listen to router traffic
    - 1.2.2.1 Router is unpatched (AND)
    - 1.2.2.2 Compromise router
    - 1.2.2.3 Guess router password

Threat Modelling Process
Step 5: Document the Threats

- Document threats by using a template

Threat description	Injection of SQL commands
Threat target	Data Access Component
Risk	
Attack techniques	Attacker appends SQL commands to user name, which is used to form a SQL query
Countermeasures	Use a regular expression to validate the user name, and use a stored procedure with parameters to access the database

- Leave Risk blank (for now)

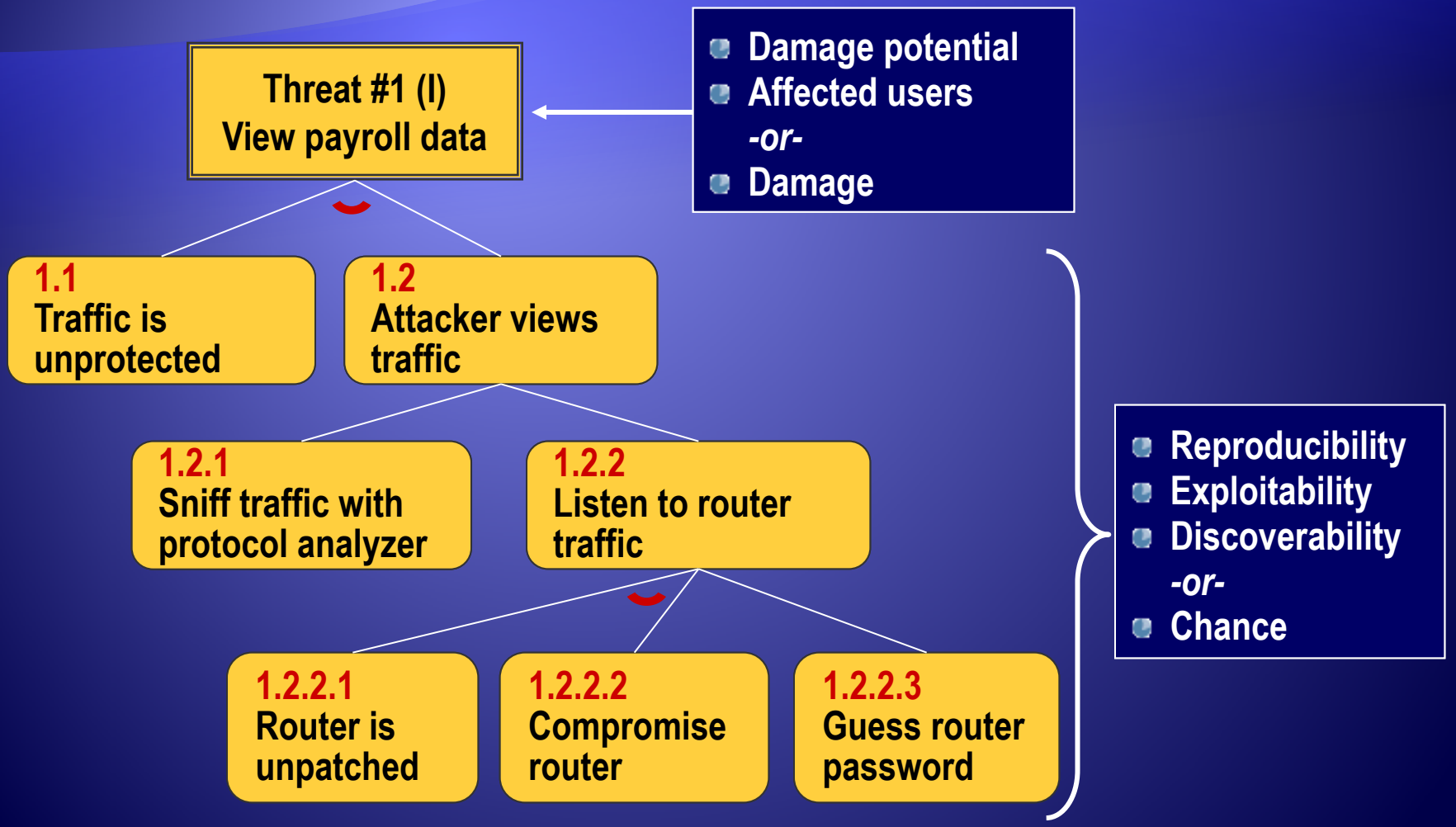
Threat Modelling Process
Step 6: Rate the Threats

- Use formula:

  Risk = Probability * Damage Potential

- Use DREAD to rate threats
  - Damage potential
  - Reproducibility
  - Exploitability
  - Affected users
  - Discoverability

Threat Modelling Process Example: Rate the Threats



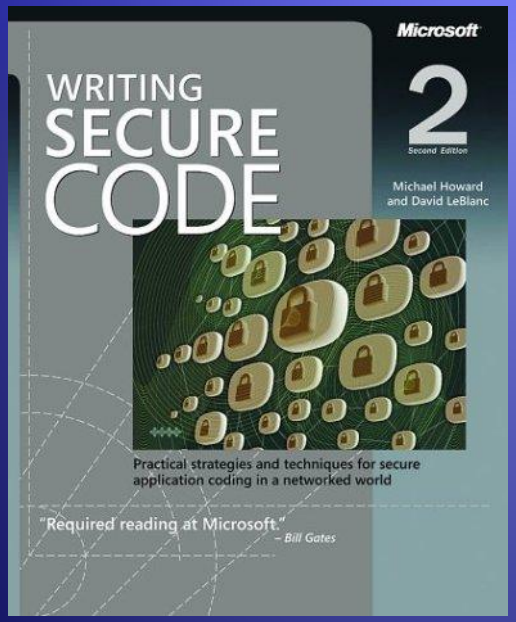
Coding to a Threat Model

- Use threat modeling to help:
  - Determine the most "insecure" portions of your application
  - Prioritize security push efforts
  - Prioritize ongoing code reviews
  - Determine the threat-mitigation techniques to employ
  - Determine data flow

Web Resources

- MSDN Security Developer Center
  - http://msdn.microsoft.com/security
- The Security Development Lifecycle
  - http://msdn.microsoft.com/security/default.aspx?pull=/library/en-us/dnsecure/html/sdl.asp?_r=1
- Top 10 Security Tips Every Developer Must Know
  - MSDN Magazine September 2002
- Application Threat Modelling
  - http://msdn.microsoft.com/security/securecode/threatmodeling/acetm/

Secure Coding Book



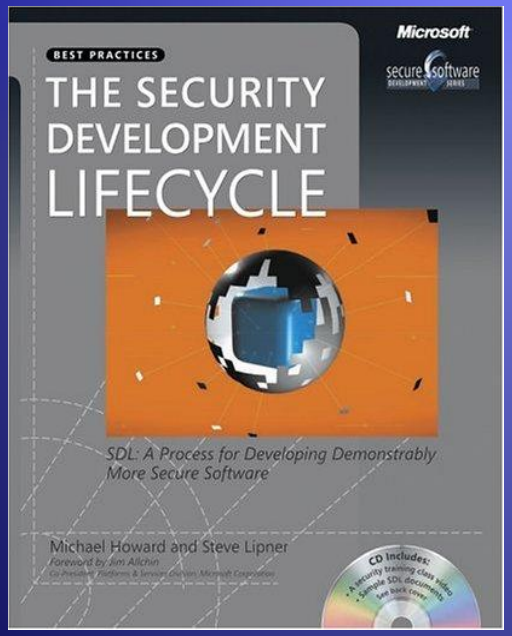
Writing Secure Code

2nd Edition

Michael Howard & David LeBlanc

- **Publisher:** Microsoft Press
- **ISBN:** 0735617228

SDL Book

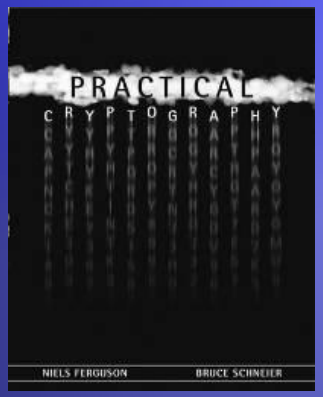


The Security Development Lifecycle

Michael Howard & Steve Lipner

- **Publisher:** Microsoft Press
- **ISBN:** 0735622140

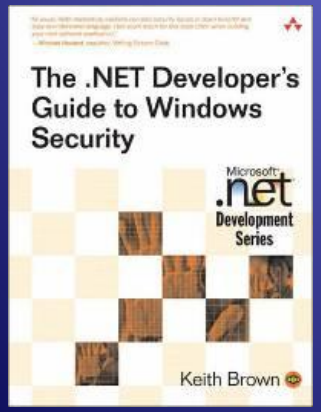
More Books



Practical Cryptography

Niels Ferguson & Bruce Schneier

- **Publisher:** John Wiley & Sons Inc.
- **ISBN: 0471223573**



The .NET Developer's Guide to Windows Security

Keith Brown

- **Publisher:** Addison Wesley
- **ISBN:** 0321228359

Microsoft®

Your potential. Our passion.™