The Security Development Lifecycle

Ch 7. Define and Follow Best Practices

Dr. Yeonjoon Lee

Roadmap

- Common Secure Design Principles
- Attack Surface Analysis and Attack Surface Reduction
- Summary

Common Secure Design Principles

- **Economy of mechanism:** Keep the code and design *simple and small*.
- **Fail-safe defaults:** The default action for any request should be to *deny* the action.
- **Complete mediation:** *Every* access to *every* protected object should be validated.
- **Open design:** As opposed to "security through obscurity", suggests that designs should *not* be secret. Kerchoff's Law. (Apple's SSL bug)

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
```

Common Secure Design Principles (cont'd)

- **Separation of privilege:** Do *not* permit an operation based on *one* condition. Examples include 2FA.
- **Least privilege:** Operate with the lowest level of privilege necessary to perform the required tasks.
- **Least common mechanism:** Minimize shared resources such as files and variables.
- **Psychological acceptability:** Easy to use?

Being Secure vs. Having Security Features

- Even the most secure design is rendered pointless by a low quality and insecure implementation, regardless of the number of security features the product employs.
- A product's security features do *not* necessarily secure the product from attack.

Attack Surface Analysis and Reduction (ASA and ASR)

- Even if your code happens to be *perfect*, it's only perfect by today's standards.
- The vulnerability research landscape is constantly evolving.
- The goal of ASA and ASR is to understand the attack surface and how to effectively reduce it to prevent an attacker from exploiting the potentially defective code.
- ASA: Process of enumerating all the interfaces and protocols and executing code.
- ASR: Process of reducing the attack surface. In other words, minimizes the code exposed to untrusted users.

Attack Surface Reduction (cont'd)

- Code with a large ***attack surface*** - that is, a large amount of code accessible to untrusted users - must be extremely high-quality code.
- ASR focuses on:
  - **Reducing** the amount of code that executes by default.
  - **Restricting** the scope of who can access the code.
  - **Restricting** the scope of which identities can access code.
  - **Reducing** the privilege of the code.

General Princip
to Reduce Attac
Surfaces:

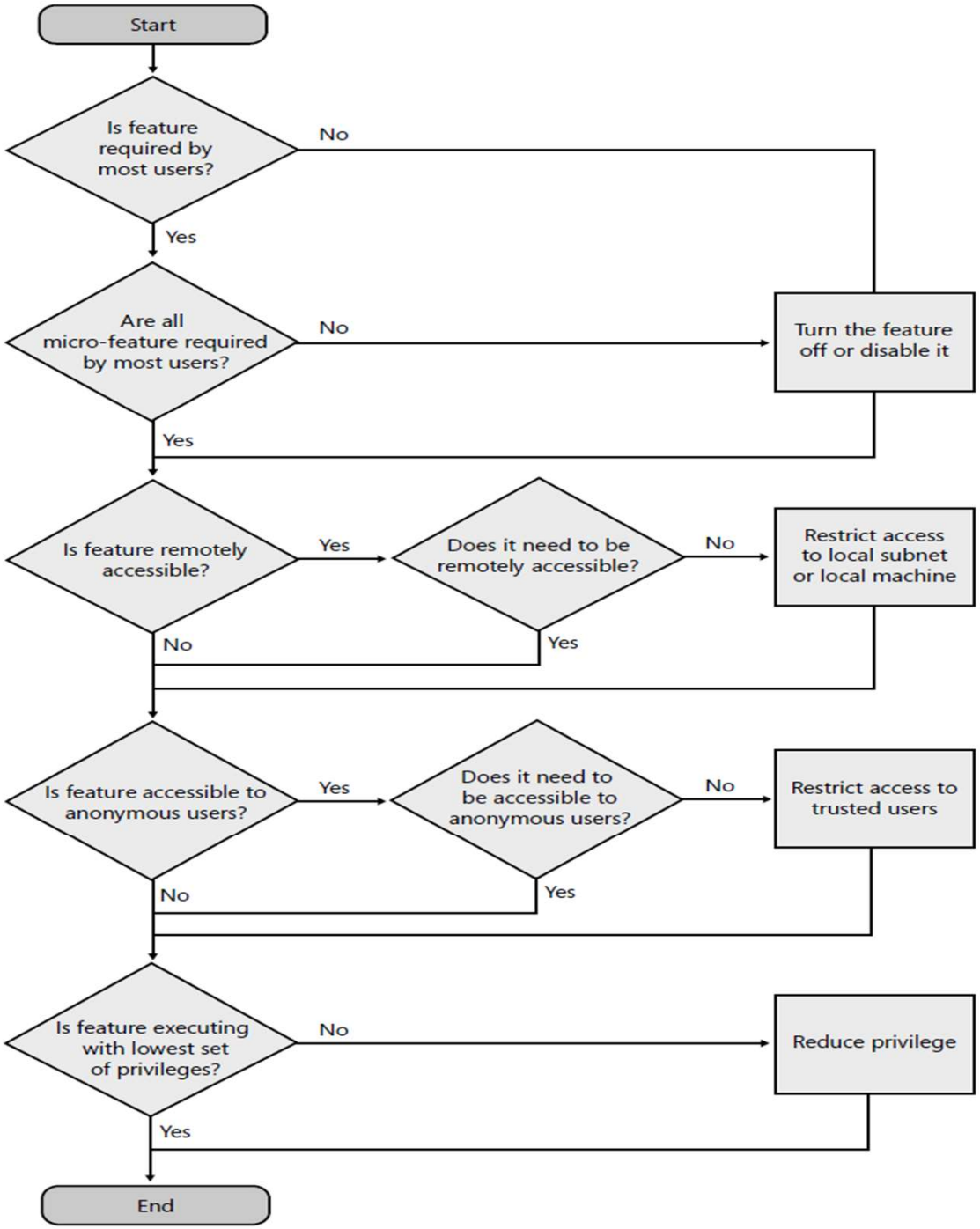

Figure 7-1 Follow these steps to reduce attack surface.

Dr. Yeonjoon Lee

Step 1: Is The Feature Really *That* Important?

- "Is this feature needed by at least 80 percent of our users?"
- If the answer is *no*, the feature should be turned off, not installed, or **disabled by default**.

Example: IIS in MS Windows Server

- [Main Feature] Internet Information Services (IIS) 6.0 Web server in Windows Server 2003 is *not* installed by default, unlike IIS 5.0 in Windows 2000, which is installed by default.
- [Micro or Sub Feature] A web server might ship with functionalities other than simple HTTP processing:
  - Various HTTP verbs (GET, POST …)
  - WebDAV requests
  - Java server requests (JSP)
  - …
- [Micro-ASR] In IIS 6.0, for example, WebDAV is *optional*. You must manually opt-in in order to use that feature.

Step 2: Who Needs Access to the Functionality and from Where?

- Code that is accessible *remotely* by *anonymous* users has a **larger** attack surface than code that is accessible only to local administrators.

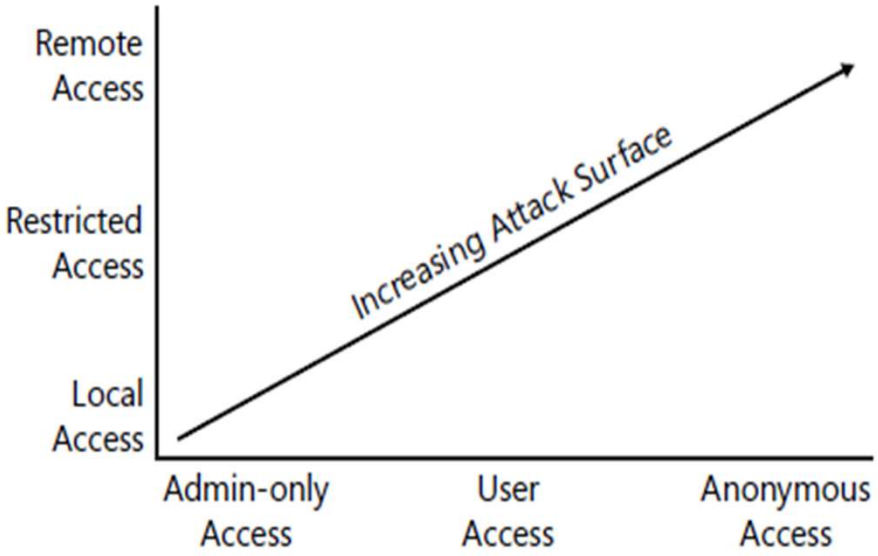

Figure 7-2 Accessibility increases attack surface.

Example: Sasser Worm

Windows Server 2003

- **Unaffected**.
- Even though the code included the security bug, the network endpoint that the worm attacks is accessible **only to administrators**.
- Developers introduce an explicit “local administrator only” check in RPC.

Windows 2000 / Windows XP

- **Affected**.
- This network interface was remotely accessible to **anonymous users**.
- No administrator check.

How?

- Consider *each* piece of your application to determine who needs to access the code by default.
- Use authentication and authorization.
  - Use authentication techniques provided at the *lowest* possible level of your system!!
  - Do *not* create your own authentication mechanisms unless absolutely necessary!!
- Restrict network accessibility
  - Use a firewall
  - By a configuration switch that defaults to the local machine, the local subnet …

Step 3: Reduce Privilege

- Processes that are exploited when running in the root context can create catastrophic failure because the exploit code will also run in the same context.
- The administrative accounts have access to all resources on the compromised computer.
- It is therefore imperative that you run code with just enough privilege to get the job done and no more.

How?

- Create an account that has just the privileges you want and have your service run under that account.
- Or run the service under a well-known account, such as Network Service and drop all the redundant privileges.
- Group membership:
  - Windows: The Administrators group is dangerous. Run the application as non-administrator.
  - Unix: Create a special group for the application.

Services and Low Privilege

- In Windows:
  - Rather than running your Windows services as Local System (SYSTEM), use less-risky service accounts such as Network Service or Local Service if possible.
- Split the application into multiple processes; only the management process runs with elevated privileges:
  - Apache httpd:
    - The 1st httpd process starts up as root. Controls other httpd processes and open HTTP ports.
    - The spawned processes run with lower-privilege accounts.
  - Microsoft IIS 6.0: similar idea.

Splitting Processes

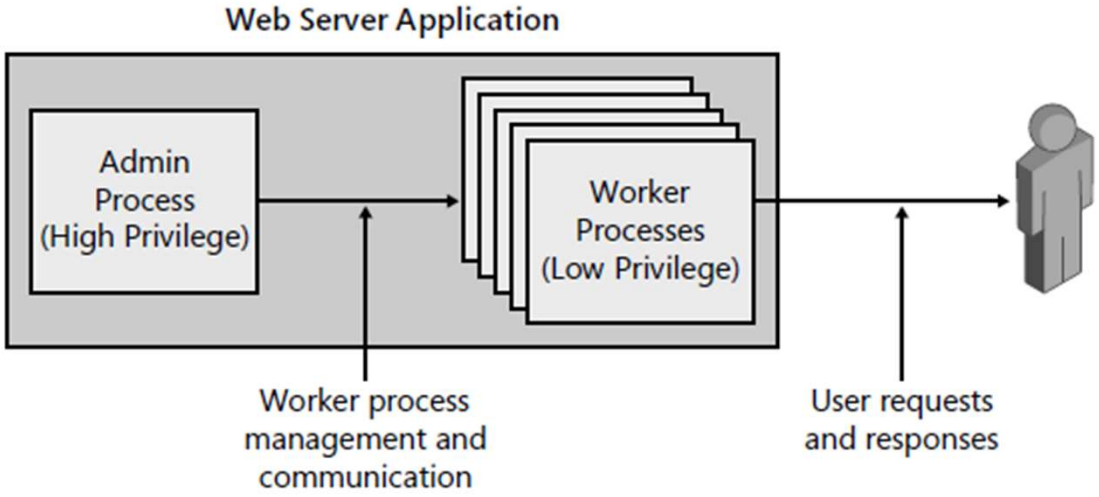

Figure 7-3 Split an application into multiple processes based on privilege. The low-privilege processes handle untrusted user requests, and the high-privilege process handles administrative tasks.

A Few Other Attack Surfaces to Consider

- UDP vs. TCP:
  - TCP performs 3-way handshake while UDP does not.
  - The source IP address can be easily spoofed when using UDP.
- Weak permissions vs. strong permissions:
  - Review your code that sets permissions to make sure untrusted users do not have access to the object.
  - Example:
    - A device driver that allows a normal user to overwrite the valid driver with rogue software. It is a local privilege elevation bug.
- A few other examples: Book page 85 ~ 86.

Application Attributes and Attack Surfaces

High Attack Surface	Low Attack Surface
Feature running by default	Feature disabled by default
Open network connection	Closed network connection
Listening for UDP and TCP traffic	Listening only for TCP traffic
Anonymous access	Authenticated user access
Authenticated user access	Administrator access
Internet access	Subnet, link-local, or site-local access
Subnet, link-local, or site-local access	Local machine access
Code running with Administrator, Local System, or root privileges	Code running with Network Service, Local Service, or custom low-privilege account
Weak object permissions	Strong object permissions
ActiveX control	.NET code
ActiveX control marked safe for scripting	ActiveX control not marked safe for scripting
Non-SiteLocked ActiveX control	SiteLocked ActiveX control

Summary

- You will *never* secure the code 100 percent.
  - You cannot predict the future (new vulnerabilities …).
  - Humans make mistakes.
- Think about security features in design phase. Implement secure designs.
- Follow the best practices can reduce attack surfaces of your product.

The Security Development Lifecycle

Ch 8. Product Risk Assessment

Dr. Yeonjoon Lee

Product Risk Assessment

Example Goals:

- What portions of the project will require threat models before release.
- What portions of the project will require security design reviews.
- What portions of the project will require penetration testing.
- The scope of fuzz testing requirements.
- What private information needs to be collected.

How?

- Security Risk Assessment.
- Privacy Impact Rating.

Security Risk Assessment (Questions)

Setup Questions:

- On which OS is your software installed?
- What permissions does it need?

Attack Surface Questions:

- Does your feature run with elevated privileges? Why?
- Does your feature listen on network sockets? Which?
- Does your feature set any firewall policy?
- Does your feature have any unauthenticated network connections?

Security Risk Assessment (Questions) – cont.

Mobile-Code Questions:

- Does your feature include ActiveX controls?
- Does your feature include any script code? If yes, what does the code do, and what languages do you use?

Security Feature–Related Questions:

- Does the application implement or use any cryptographic mechanism?
- Does the application implement any security mechanisms such as authentication or authorization?

Security Risk Assessment (Questions) – cont.

General Questions:

- Has this product had serious security bugs in the past?
- Does your application parse files?
- Does your application query a database?
- What components can download and execute code?
- Does your application have user-mode and kernel-mode components?

Security Risk Assessment (Analyzing the Questions)

General Rules

- Every method on ActiveX control must be reviewed
- New product requires a thorough security design review
- Sample code must meet the same quality standards as shipping code
- If an application parses files or network traffic, the application is subject to the SDL fuzzing requirements

Security Risk Assessment (Analyzing the Questions) – cont.

- The following cases must be threat modelled:
  - Application with networking interface
  - Application with kernel-mode and user-mode
  - Applications where non-administrators interact with higher-privileged processes
  - Applications that are security features

Privacy Impact Rating

Important terminology:

- **Anonymous Data**: Any user data that is not unique or tied to a specific person which cannot be traced back
- **Personally Identifiable Information (PII)**: Any user data that uniquely identifies a user (e.g., name, address, phone number, e-mail address, etc.)
- **Sensitive PII**: Any user data that identifies an individual and could facilitate identity theft or fraud (e.g., identification number, credit card numbers, bank account numbers, biometric information, etc.)

Privacy Ranking 1

If any of the following statements are true, your application requires the highest privacy:

- The application stores PII or transfers PII to developer
- The application is targeted at children or could be deemed attractive to children
- The application continuously monitors the user
- The application installs new software or changes the user's file-type associations

Privacy Ranking 2,3

Privacy Ranking 2:

If your application transfers anonymous data to the software developer or to a third party

Privacy Ranking 3:

If the application exhibits none of the behaviors in privacy rankings 1 and 2

The Security Development Lifecycle

Ch 9. Risk Analysis

Dr. Yeonjoon Lee

Roadmap

- Threat Modeling Basics
- The Threat-Modeling Process
  - Using a Threat Model to Aid Code Review
  - Using a Threat Model to Aid Testing
- Key Success Factors and Metrics

Threat-Modeling Artifacts

- The main output of the threat-modeling process is a document (or documents) that
  - describes *background information* about the application (see next slide)
  - defines the *high-level application model* often by using the following: data flow diagrams (DFDs); a list of assets that require protection; threats to the system ranked by risk; and, optionally, a list of mitigations

Threat-Modeling Artifacts (cont'd)

- Background information includes:
  - **Use scenarios**
    - Deployment configurations and broad customer uses
  - **External dependencies**
    - Products, components, or services the system relies on
  - **Security assumptions**
    - Assumptions you make about the security services offered by other components
  - **External security notes**
    - Information useful to your product's end user or administrator to operate the system securely

What to Model

- First, consider the *trust boundaries* of your application, and model all the components inside that trust boundary
- Next, look outside the boundaries to determine what is really part of your application

Building the Threat Model

- Model-building process follows the following steps:
  - Prepare
    - System designers take the lead in preparing, with input from the development team, to build a DFD (Data Flow Diagram).
    - The resulting artifact is sent to other team members for review before the core threat-modeling analysis process begins.
  - Analyze
  - Determine mitigations

Building the Threat Model (cont'd)

- Model-building process follows the following steps:
  - Prepare
  - Analyze
    - All threats are uncovered through the analysis process and are added to the threat model document.
    - More people will be included in the process.
    - Also remember that at this stage you should discuss only threats, not mitigations.
  - Determine mitigations

Building the Threat Model (cont'd)

- Model-building process follows the following steps:
  - Prepare
  - Analyze
  - Determine mitigations
    - More of the product team is involved in identifying mitigations.
    - This step is performed once the threat model is basically complete.
    - The team considers the model to determine the appropriate remedies to the threats.

The Threat-Modeling Process

- 1. Define use scenarios.
- 2. Gather a list of external dependencies.
- 3. Define security assumptions.
- 4. Create external security notes.
- 5. Create one or more DFDs of the application being modeled.
- 6. Determine threat types.
- 7. Identify the threats to the system.
- 8. Determine risk.
- 9. Plan mitigations.

The Threat-Modeling Process (cont'd)

- 1. Define Use Scenarios
  - ➢ the team needs to determine which key threat scenarios are within scope
  - ➢ should also consider the insider-threat scenario
  - ➢ also include other common, but not security-related, scenarios such as the type of customer you expect to use your software

The Threat-Modeling Process (cont'd)

- 2. Gather a List of External Dependencies
  - ➢ It's important that you document all the other code your application depends on, e.g. operating system, database, Web server, or high-level application framework
  - ➢ You should also consider the default system-hardening configuration

The Threat-Modeling Process (cont'd)

- 3. Define Security Assumptions
  - If you make inaccurate security assumptions about the environment in which the application resides, your application might be rendered utterly insecure
  - If your application stores encryption keys, assume that the operating system will protect the keys correctly
    - For Microsoft Windows XP and later, this might be true if you store the keys using the data protection API
    - For Linux here is no such service, so the assumption is incorrect

The Threat-Modeling Process (cont'd)

- 4. Create External Security Notes
  - Users and other application designers who interact with your product can use external security notes to understand your application's security boundaries and how they can maintain security when using your application.

The Threat-Modeling Process (cont'd)

- 5. Create One or More DFDs of the Application Being Modeled
  - The highest-level DFD is the context diagram, which shows the system under development at the center and the external entities that interact with the system.

The Threat-Modeling Process (cont'd)

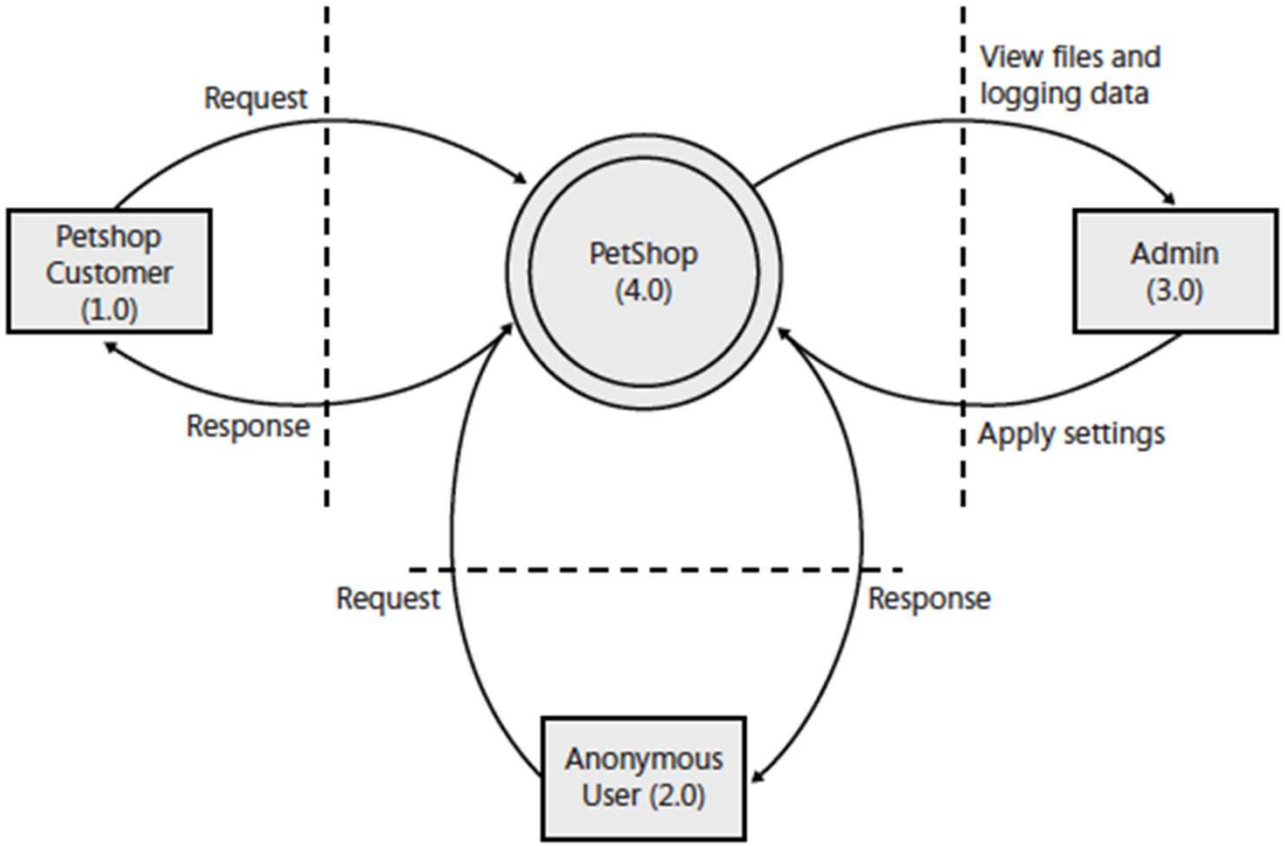

Figure 9-3 Context diagram for Pet Shop 4.0.

The Threat-Modeling Process (cont'd)

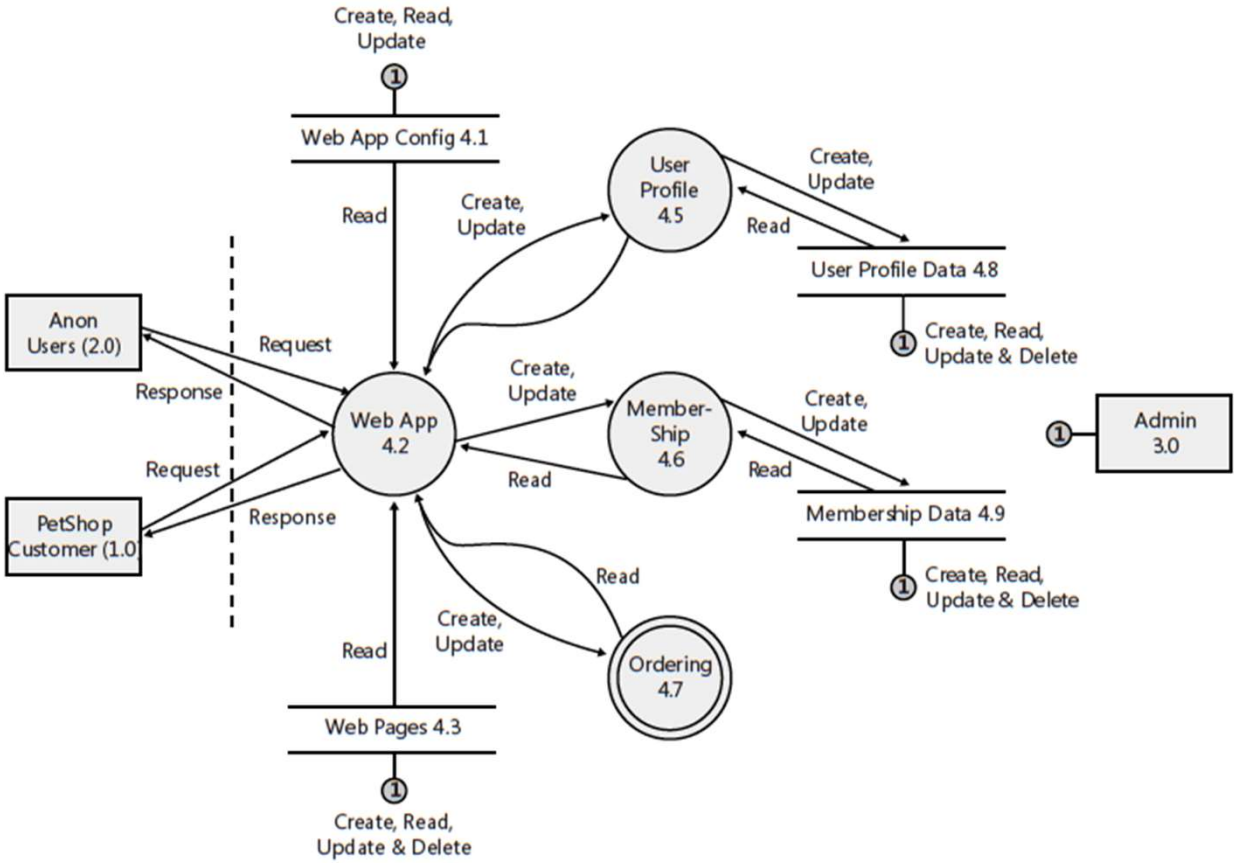

Figure 9-4 The level-0 DFD for Pet Shop 4.0.

The Threat-Modeling Process (cont'd)

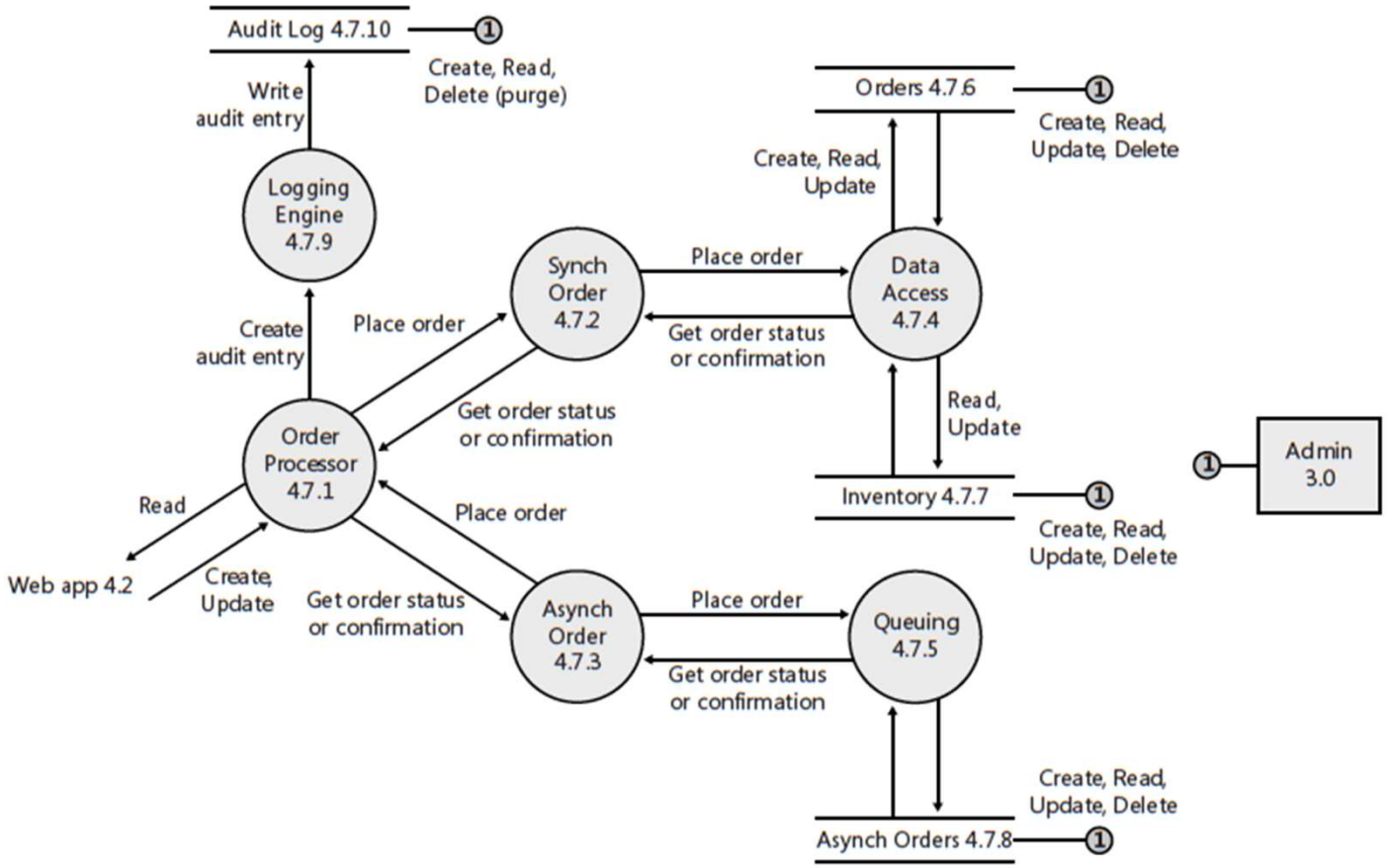

Figure 9-5 The level-1 DFD for Pet Shop 4.0.

The Threat-Modeling Process (cont'd)

- 6. Determine Threat Types
  - Microsoft uses a threat taxonomy called STRIDE to identify various threat types
  - STRIDE looks at threats from an attacker's perspective

The Threat-Modeling Process (cont'd)

- 6. Determine Threat Types (STRIDE)
  - **S**poofing Identity
    - Spoofing allow an attacker to pose as something or somebody else
  - **T**ampering
    - Tampering threats involve malicious modification of data or code.
  - **R**epudiation
    - An attacker deny to have performed an action that other parties can neither confirm nor contradict

The Threat-Modeling Process (cont'd)

- 6. Determine Threat Types
  - **I**nformation Disclosure
    - Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it.
  - **D**enial of Service
    - DoS attacks deny or degrade service to valid users
    - You must protect against certain types of DoS threats simply to improve system availability and reliability
  - **E**levation of Privilege
    - EoP threats often occur when a user gains increased capability

The Threat-Modeling Process (cont'd)

- 7. Identify Threats to the System
  - Once the DFD is done, you need to list all the DFD elements because you need to protect these elements
  - Apply a process called *reduction* to reduce the number of entities you will analyze
  - Once the list of DFD elements is complete, you can apply STRIDE to each of the elements in the list by following the mapping of STRIDE categories to DFD element types

The Threat-Modeling Process (cont'd)

- 8. Determine Risk
  - Historically, security specialists used numeric calculations to determine risk
    - Problem is that the numbers can be very subjective
  - Others tried to determine the chance of an attack
    - No idea about what the chance of attack really is
  - Microsoft has created a *bug bar* that defines level of risk; they use the term "risk level" to indicate overall risk; risk level 1 is highest and risk level 4 is the lowest

The Threat-Modeling Process (cont'd)

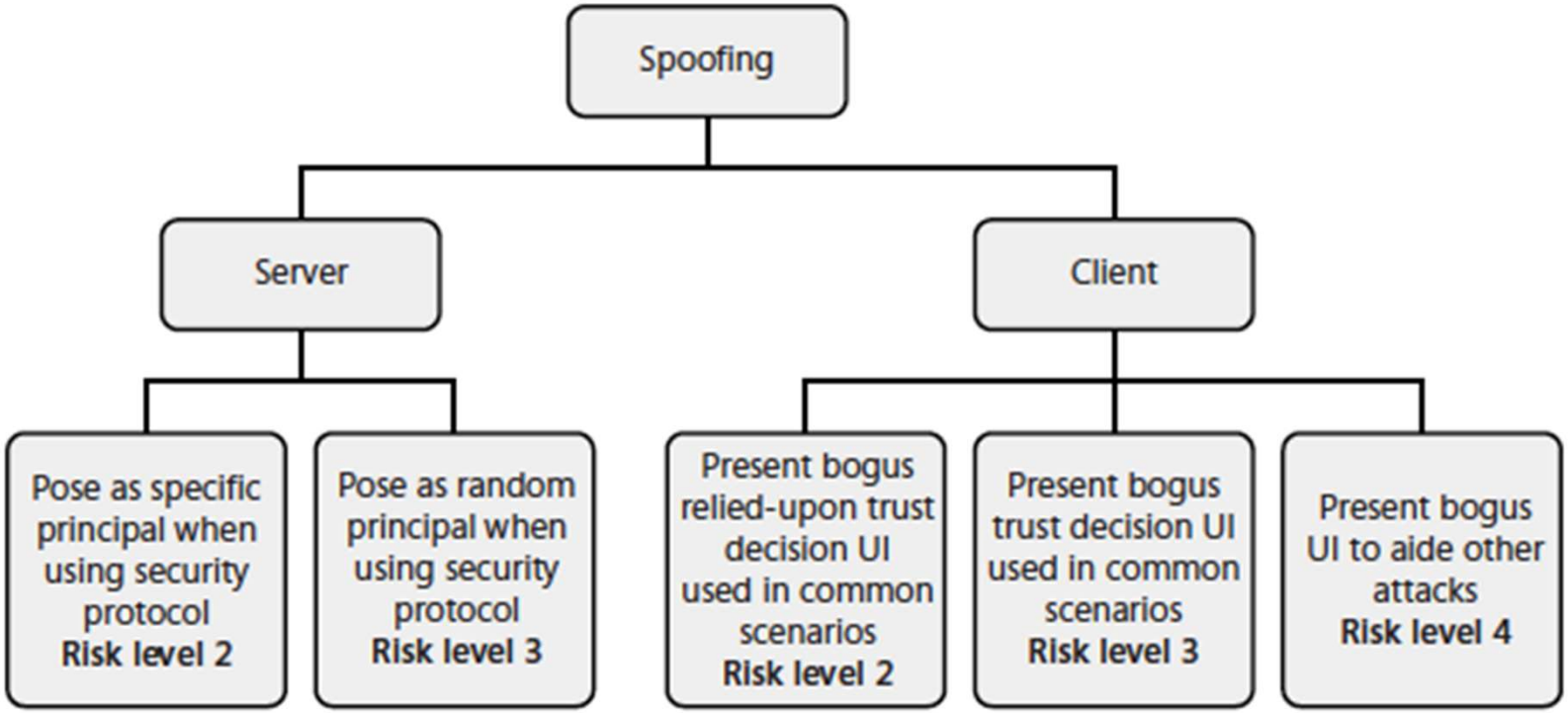

Figure 9-6 Spoofing threats risk ranking.

The Threat-Modeling Process (cont'd)

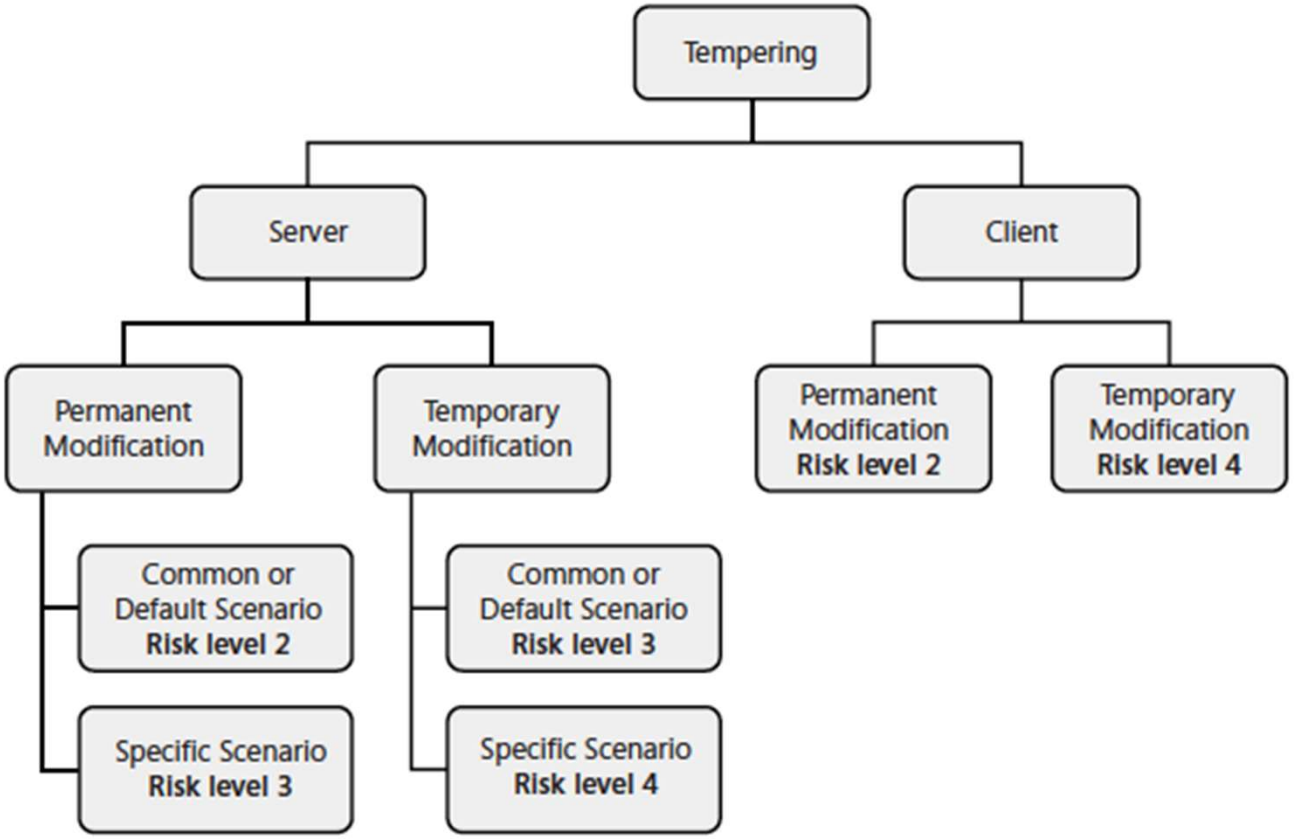

Figure 9-7 Tampering threats risk ranking.

The Threat-Modeling Process (cont'd)

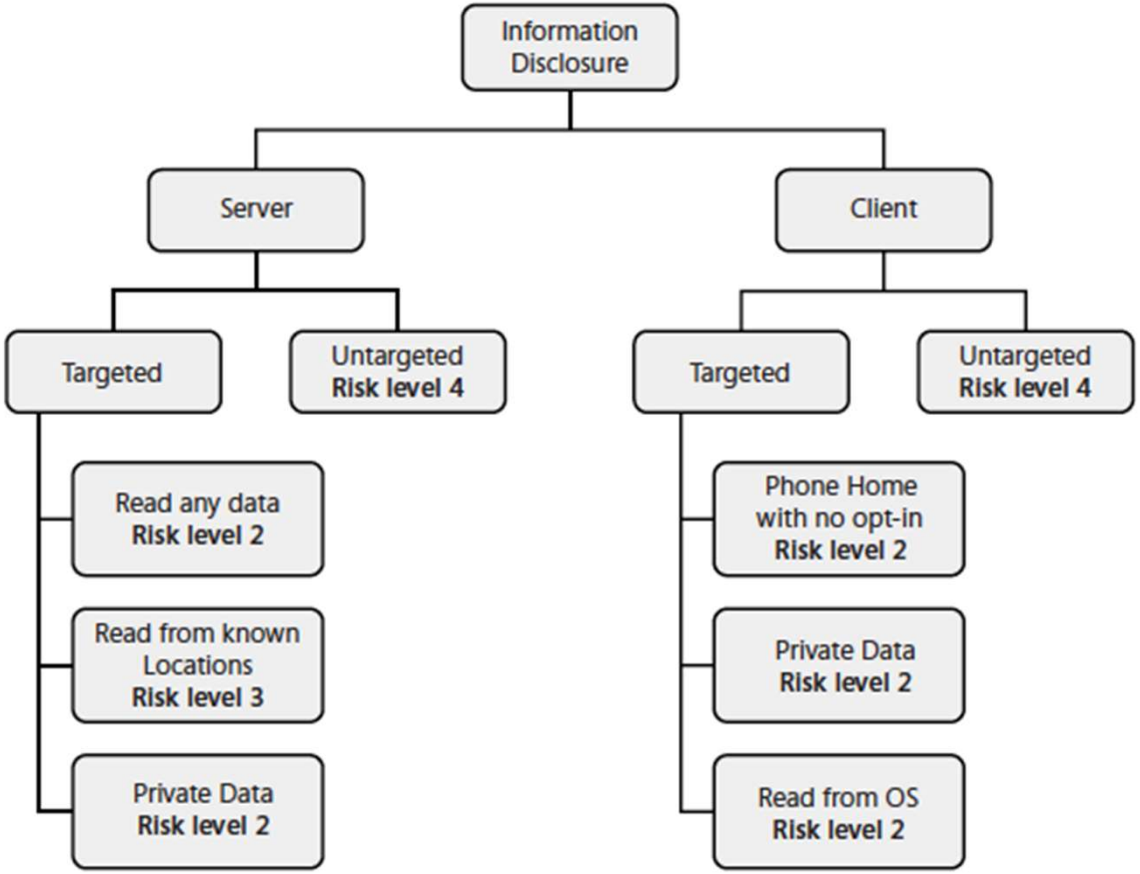

Figure 9-8 Information disclosure threats risk ranking.

The Threat-Modeling Process (cont'd)

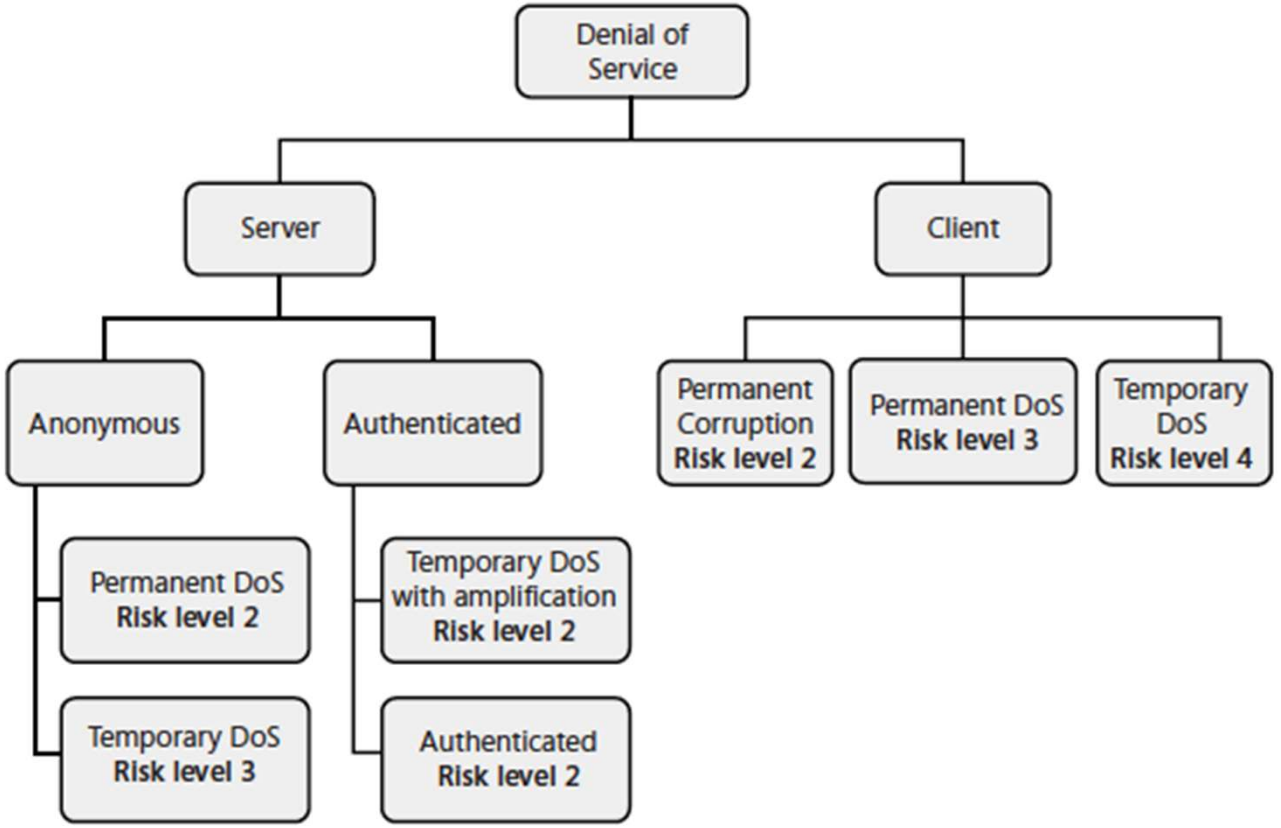

Figure 9-9 DoS threats risk ranking.

The Threat-Modeling Process (cont'd)

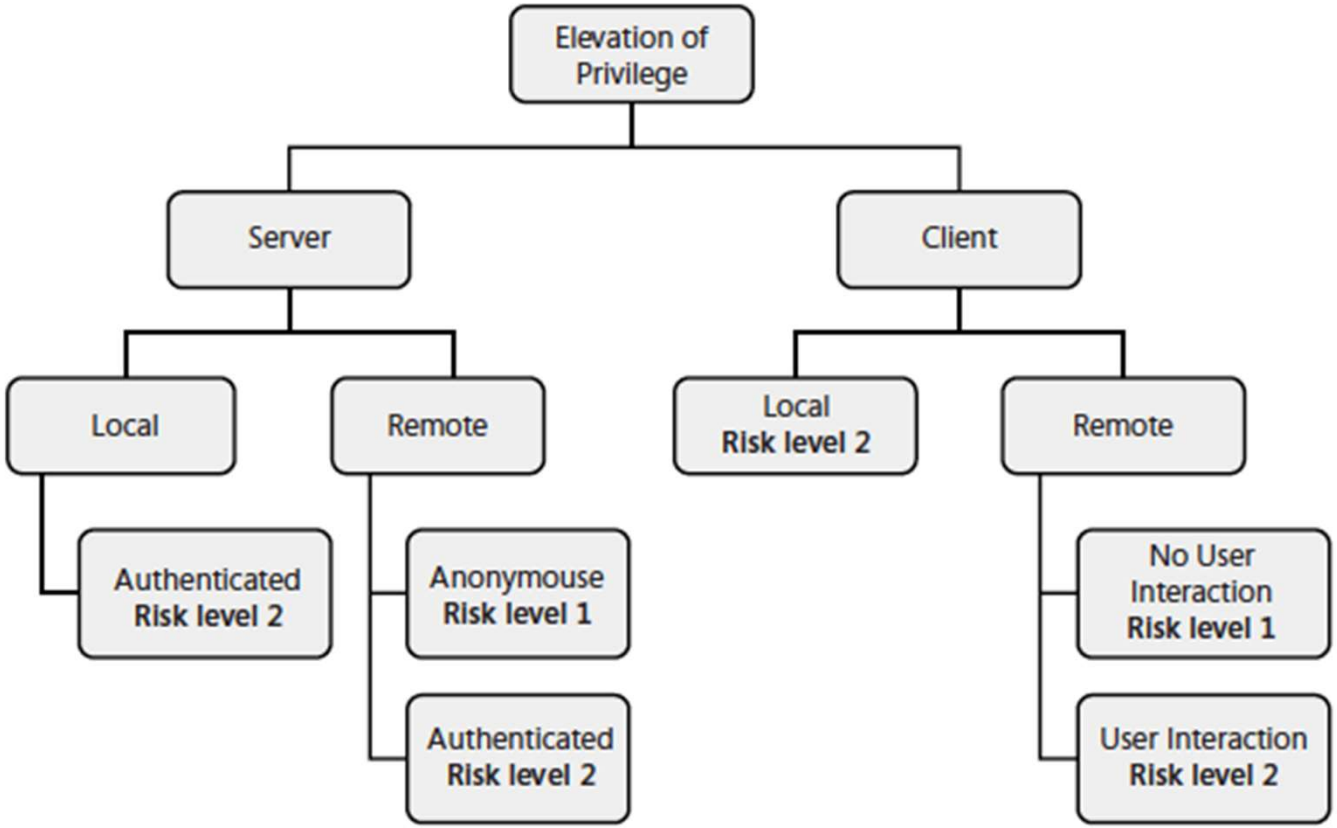

Figure 9-10 EoP threats risk ranking.

The Threat-Modeling Process (cont'd)

- 9. Plan Mitigations

  - Often referred to as countermeasures or defenses, mitigations reduce or eliminate the risk of a threat

  - Mitigation strategies:

    - ➢ Do nothing
      - For low-risk threats, doing nothing could be a valid strategy
    - ➢ Remove the feature
      - a balancing act between user features and potential security risks
    - ➢ Turn off the feature
      - Consider this only as a defense-in-depth strategy

The Threat-Modeling Process (cont'd)

- 9. Plan Mitigations
  - Mitigation strategies:
    - ➢ Warn the user
      - most users, especially non-technical users, make poor trust decisions
    - ➢ Counter the threat with technology (most common / see below)

Table 9-8 Mitigation Techniques Based on STRIDE Threat Type

Threat Type	Mitigation Technique
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation services
Information disclosure	Confidentiality
DoS	Availability
EoP	Authorization

Using a Threat Model to Aid Code Review

- One of the deliverables from the threat-modeling process is a list of *entry points* to the system. This is what the context diagram shows
- When it comes to reviewing the code for security bugs, it's imperative that you review all code that is *remotely* and *anonymously accessible* before reviewing other code

Using a Threat Model to Aid Testing

- Specific threat types (spoofing and tampering, for example) have specific mitigation techniques. These techniques can also be attacked.
- Determine how best to build attacks or perform penetration testing by looking at the relevant threats

Key Success Factors and Metrics

- Determining a good threat model is rather subjective
- Use metrics to separate the good from the not-so-good threat models
- At a minimum, threat models should be rated “OK” and components that are to be penetration tested should be “Good” or better (see next slides about the guidelines)

Key Success Factors and Metrics (cont'd)

Table 9-12 Threat-Model Quality Guidelines

Rating	Comments
No threat model (0)	■ No threat model is in place—this is simply not acceptable because it indicates that no threats are being considered.
Not acceptable (1)	■ Threat model is clearly out of date if: ❑ Current design is significantly different from that defined in the threat model. ■ –Or– ❑ Date in document shows that it is older than 12 months.
OK (2)	■ A data flow diagram or a list of the following exists: ❑ Assets (processes, data stores, data flows, external entities) ❑ Users ❑ Trust boundaries (machine to machine, user to kernel and vice versa, high privilege to low privilege and vice versa) ■ At least one threat is detailed for each software asset. ■ Mitigations are provided for all risk level 1, 2, and 3 threats. ■ Model is current.

Key Success Factors and Metrics (cont'd)

Table 9-12 Threat-Model Quality Guidelines

Rating	Comments
Good (3)	■ Threat model meets all definitions of "OK" threat models. ■ Anonymous, authenticated, local, and remote users are all shown on the DFD. ■ All S, T, I, and E threats have been identified and classified as either mitigated or accepted.
Excellent (4)	■ Threat model meets all definitions of "Good" threat models. ■ All STRIDE threats have been identified and have mitigations, external security notes, or dependencies acknowledged. ■ Mitigations have been identified for each threat. ■ External security notes include a plan to create customer-facing documents (from the external security notes) that explain how to use the technology safely and what the tradeoffs are.

Summary

- Threat modeling is critically important to helping build secure software
  - the cornerstone to understanding how your product could be attacked and how to defend it
- The process is also a great way to determine the overall security health of a software development team
  - security-savvy teams are more in tune with the threats to their code
- Systematically uncover threats to the application, rank the risk of each threat, and determine appropriate mitigations
  - help to perform code reviews and build penetration tests.