PREV NEXT

The Threat-Modeling Process

Although the high-level steps involved in creating a threat model may seem numerous, many of the elements require little security expertise and are virtually rote. The steps are as follows:

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.

1. Define Use Scenarios

At this stage of the process, the team needs to determine which key threat scenarios are within scope. For example, if you create a mobile or small device, you'll probably want to cover the stolen-device scenario. However, you may determine that the device stores no sensitive data whatsoever, so the risk associated with a stolen device is low. In this situation, you should explicitly state the reasoning behind your decision. If you don't, someone years from now is bound to ask why you did not cover the stolen-device scenario! If you later decide to store sensitive data on the device, you'll need to revisit the model.

You should also consider the insider-threat scenarioshould your product protect against attackers who work for your company? If so, the threats you need to consider will be quite different from those that you can expect from external threat agents. Never lose track of who you are up against.

Also include other common, but not security-related, scenarios such as the type of customer you expect to use your software. An application designed solely for administration purposes (for example, a disk analysis tool) has a different threat profile than a product that is primarily accessed by anonymous users (for example, a Web or e-mail server).

Caution

Do not confuse use scenarios with UML Use Cases; the terms sound similar, but they are not the same.

2. Gather a List of External Dependencies

Your application is not self-sufficient: it runs on operating systems and might use a database, Web server, or high-level application framework. It's important that you document all the other code your application depends on. For example, the Internet-based pet store application, Pet Shop 4.0, might depend on the following:

- A Web-based client using Microsoft Internet Explorer 6.0 or later and FireFox 1.5 or later.
- Microsoft Windows Server 2003 and Solaris 10 servers
- On the server only, Microsoft SQL Server 2005 (on Windows Server 2003) and Oracle 10g (on Solaris 10)
- On the server only, Microsoft Message Queue 2.0
- Microsoft .NET Framework 2.0 and common language runtime 2.0 (server only)

You should also consider the default system-hardening configuration. For example, in the case of Windows Server 2003, you might require only the default hardened version of the operating system to run. Or perhaps you might need to loosen some of the security settings in the system. If you do this, you must inform your users.

3. Define Security Assumptions

This is a critically important section because if you make inaccurate security assumptions about the environment in which the application resides, your application might be rendered utterly insecure. For example, if your application stores encryption keys, a design requirement might be that you rely on the underlying operating system to protect the keys, so the assumption is that the operating system will protect the keys correctly. Let's analyze this assumption: in the case of Microsoft Windows XP and later, this might be true if you store the keys using the data protection API (DPAPI). However, in the case of Linux (as of the 2.6 kernel), there is no such service, so the assumption is incorrect, and you shouldn't store encryption keys in plaintext unless you have other viable defenses. For the record, most files holding sensitive data are stored in Linux by using a permission that allows access to only the most trusted user, the root account. But this storage option might not be enough; you might want to allow only a specific user access to his or her sensitive data, not the administrator and not the root account.

But trying to restrict access to only valid users and not administrators is interesting; we have defined some external security notes about the role of the root account.

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 prior section mentions that sensitive files in Linux are often protected by a permission that allows only root to access them. This means that root, like the administrator and SYSTEM accounts in Windows, is all-powerful and can usually read or manipulate any file or setting in the operating system. Your users should know this.

In some scenarios, this might not be the case if you are using SELinux, which can enforce mandatory access control or, in the case of tampering rather than disclosure, Microsoft Windows Vista with its mandatory integrity control might suffice. These are both examples of security technologies and dependencies that you assume behave as advertised.

As you can see, there is a tight relationship among external security notes, security assumptions, and dependencies. The following Pet Shop example shows how external dependencies, security assumptions, and external security information might relate.

Pet Shop 4.0 External Dependencies

We expect and rely on the following components within the system. You should also outline specific version numbers.

- Client
  - Clients running Internet Explorer 6.0 or later or FireFox 1.5 or later.
- Servers
  - Windows Server 2003 SP1
  - Microsoft Internet Information Services (IIS) 6.0 (Web server computers)
  - Microsoft ASP.NET 2.0 and .NET Framework 2.0 (Web server computers)
  - Microsoft SQL Server 2000, Microsoft SQL Server 2005, or Oracle 10g (database server computers)
  - Windows Server 2003 Terminal Services (all servers)
  - Microsoft Message Queue 2.0 (Web server computers)
  - Microsoft Distributed Transaction Coordinator (MSDTC) (all computers)

Pet Shop 4.0 Security Assumptions

Security assumptions are the guarantees you expect from the external dependencies:

- No sensitive data is deliberately persisted on the client, but sensitive data is sent over SSL/TLS connections, and some browsers might locally cache data sent over these connections.
- DPAPI is used on the server to protect sensitive connection strings and encryption keys for non-administrators.
- The database server holds authentication information.
- The database server adequately protects authentication data by using database server authorization technology and, potentially, encryption.

- IIS 6.0 and ASP.NET enforce authentication correctly.
- Web service security configuration is held in Web.config files and can be manipulated only by valid administrators. This rule is enforced through operating system access control lists (ACLs).
- The server application setup program correctly configures the ACL for the Web.config file.
- Only valid admins administer any server by using Terminal Services or physically accessing the server when needed.
- Only valid admins have physical access to the Web and database servers.
- Browsing the Internet, reading e-mail messages, and using peer-to-peer or instant messaging applications on the servers is expressly prohibited.
- Databases are configured to use their native authentication protocols rather than operating system authentication protocols. For example, SQL Server uses Standard authentication and not Windows authentication. This is done for two main reasons. First, most users are Internet-based users, not Windows users. The second reason is performancenative authentication schemes are faster.
- The connection information for each database is stored in the application's Web.config file and protected by using the Protected Configuration option, which uses DPAPI.

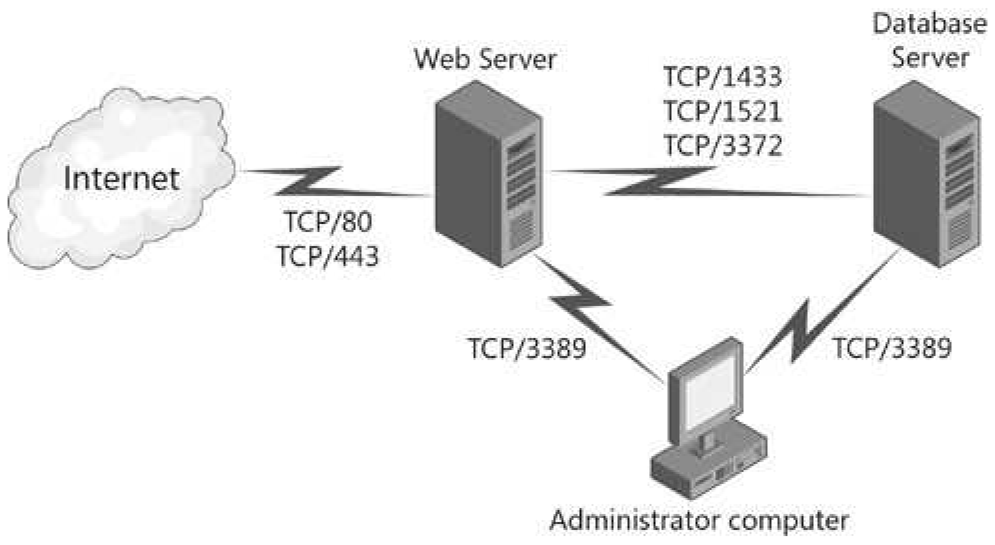
Pet Shop 4.0 External Security Information

The threat model must explain security-relevant information that end users could employ to secure their systems or, in some cases, understand the security ramifications of enabling certain functionality. Here's the list of external security information for our running example:

- Admins can change any setting in the system, including the Web service.
- The only ports open are TCP/3389 (for administration accessible only to other administration computers), TCP/80 (for HTTP traffic accessible from the Internet), TCP/443 (for HTTPS traffic accessible from the Internet), TCP/1433 and TCP/1521 (for database access that is accessible only by the Web server and by administration computers), and TCP/3372 (for MSDTC, which is accessible by all computers involved in order-processing transactionsthese are usually just the Web server computers and database server computers).

The network diagram in Figure 9-1 shows the port relationship among computers in the application.

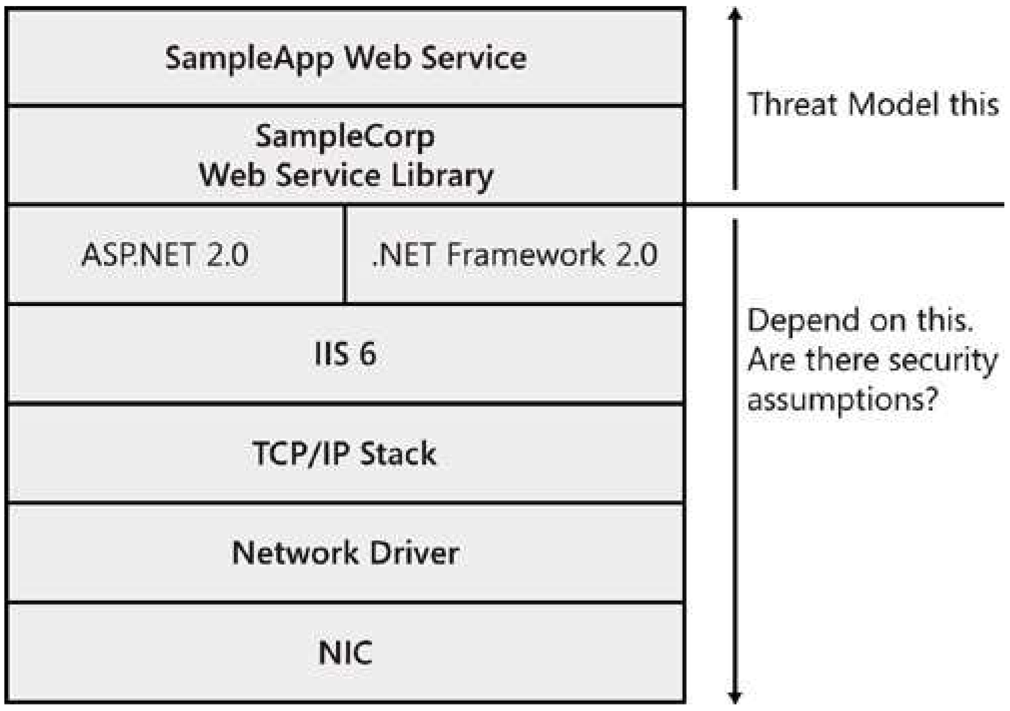
Figure 9-1. Port usage in the Pet Shop 4.0 application.



What Is Modeled and What Do You Depend On?

What follows is a best practice that can help uncover the boundary between something you control and can threat-model and something you can't control but depend on, which might have security assumptions. This will probably be iterative because you need to understand where to interface with components you don't control. Let's say you're building a Web application that sits on top of some Web class helper libraries you wrote, which in turn use a framework library, which sits on top of a Web server, which sits on top of a TCP/IP stack, which sits on top of a network driver, which sits on top of a network card. Figure 9-2 schematically represents this scenario. When you put a line just below whatever you control, everything above the line should be in your threat model, and everything below it is something you depend on.

Figure 9-2. Drawing the line between what you threat-model and what you depend on.



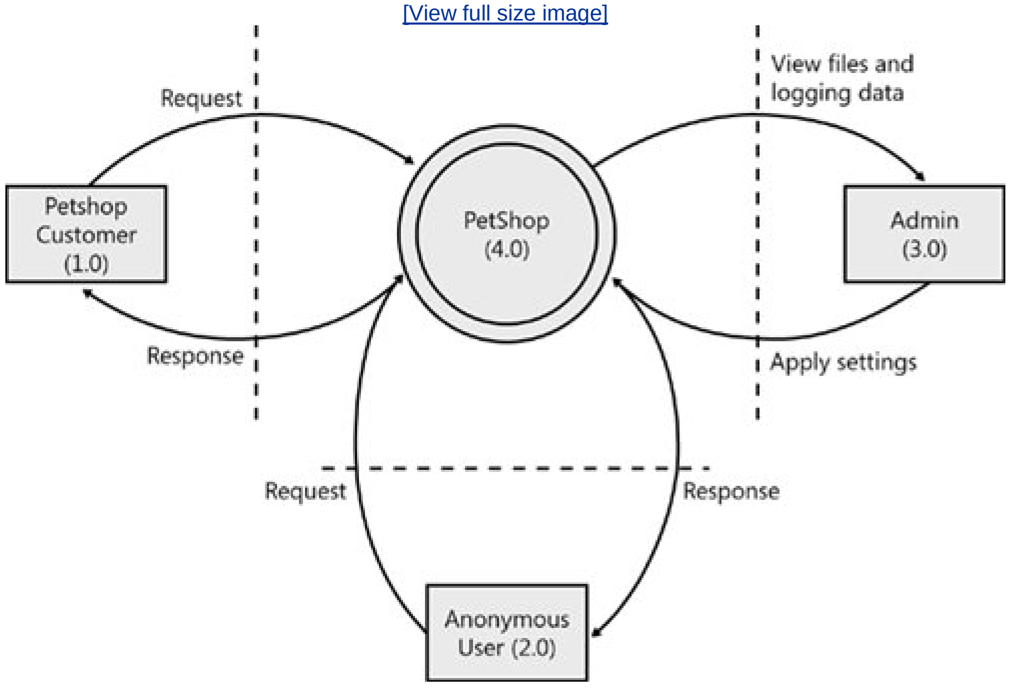
5. Create One or More DFDs of the Application Being Modeled

The next stage is creating DFDs for the application. It's critical that you get this right because if the DFD is wrong, the rest of the threat-modeling process is wrong. We won't explain the DFD process in great detail here because there is plenty of good literature available on the subject (Kozar 1997, Sauter 2002, Drewry 2005, Ambler 2006, DFD 2006, Yourdon 2006). However, we'll cover enough to give you a good overview of the process.

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 context diagram helps you understand who interacts with your code. Figure 9-3 shows a context diagram for Pet Shop 4.0.

Figure 9-3. Context diagram for Pet Shop 4.0.

[View full size image]



The shapes listed in Table 9-2 are used when building DFDs.

Table 9-2. DFD Element Types

Shape	DFD Element Type	Description
<double circle>	Complex process (also called a multiprocess)	A logical representation of a process that performs many distinct operations. Examples include a service or daemon, an assembly, a Win32 .exe file that hosts many dynamic-link libraries (DLLs).
<circle>	Process	A logical representation of a process that performs one discrete task. Some DFD references use a rounded rectangle to represent a process.

<rectangle>	External entity (also interactor)	Someone or something that drives your application but that your application cannot control. Examples include system users, asynchronous events, and external processes.
<parallel lines>	Data store	Persistent data storage such as files and databases; could also include cached information. Some DFD references use an open-ended rectangle to represent a data store.
<arrowed line>	Data flow	Means by which data moves around the system. Examples include networking communications, shared memory, and function calls.
<dotted line>	Privilege boundary (also trust boundary)	Specific to threat modeling, privilege boundaries delineate data moving from low to high trust and vice versa. Examples are machine-to-machine boundaries, process boundaries (where, for example, a low-privilege user communicates with a high-privilege process), and the line between kernel-mode and user-mode code.

As you can see in the context diagram in Figure 9-3, one central complex process is the entire Pet Shop 4.0 application itself. The only external entry points are to anonymous users, Pet Shop customers, and administrators.

Value of Privilege or Trust Boundaries

Trust boundaries are demarcation points in the application that show where data moves from lower privilege to higher privilege. These boundaries can help pinpoint areas in the application where data must be analyzed for correctness, but they can also be the site of sensitive data leaks. For example, an anonymous user creates a request that is sent to a higher-privilege process for consumption. Because the data moves from low privilege to a higher privilege, the request must be vetted for correctness. Any code on a boundary like this must be human-reviewed for correctness. This is especially true if the trust delta is large. For example, an anonymous user interacting with code running as admin, root, or system constitutes a very large privilege delta, whereas an administrator interacting with the same code constitutes a substantially lower privilege delta.

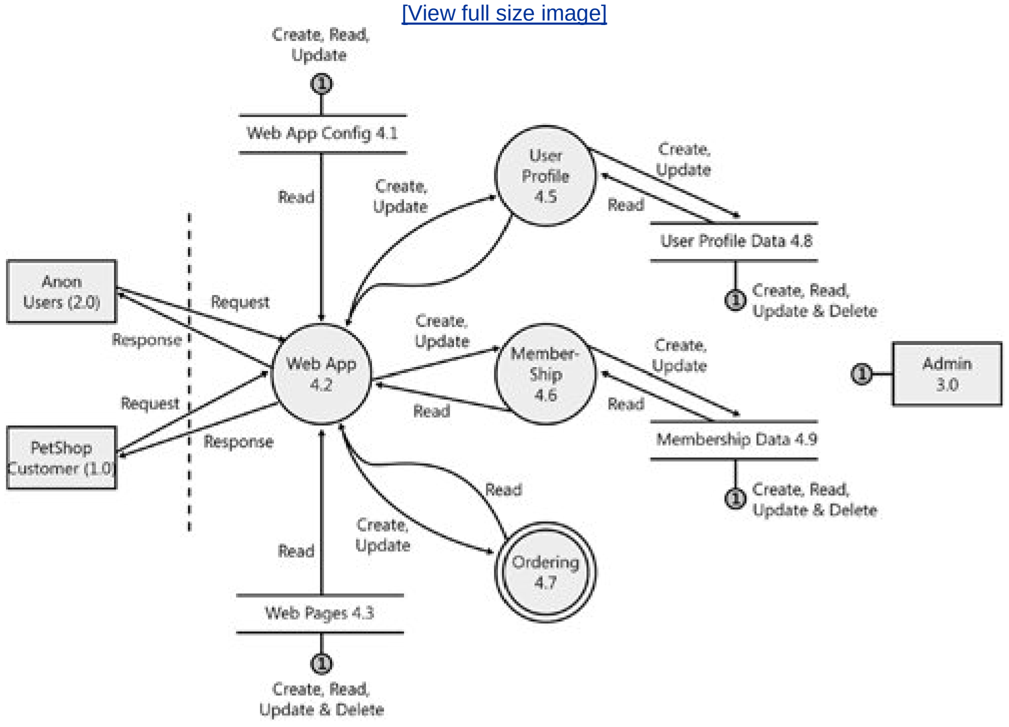
But there is more to this. Because data moving from high to low privilege must not leak sensitive, private, or confidential data, you must analyze the data traveling from the high-privilege process to make sure error messages and the like do not leak enough to aid an attacker.

Notice how all the components of the context diagram in Figure 9-3 are numbered, and each data flow is associated with a verb or verb/noun. Adding verbs and possibly nouns to a data flow helps provide context for the tasks you expect the user to perform. For some data flows, you can use create, read, update, and delete (CRUD) nomenclature. This nomenclature is fine for most data flows, but it's not very descriptive because you often need to include a noun to describe what is being manipulated.

The next step is to look at the context diagram, drill down into the complex processes, and create the next diagram, which is called the level-0 DFD. The context diagram for Pet Shop 4.0 has only one complex process: element 4.0. Figure 9-4 shows the level-0 DFDthe view of the application when you drill inside the Pet Shop 4.0 complex process.

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

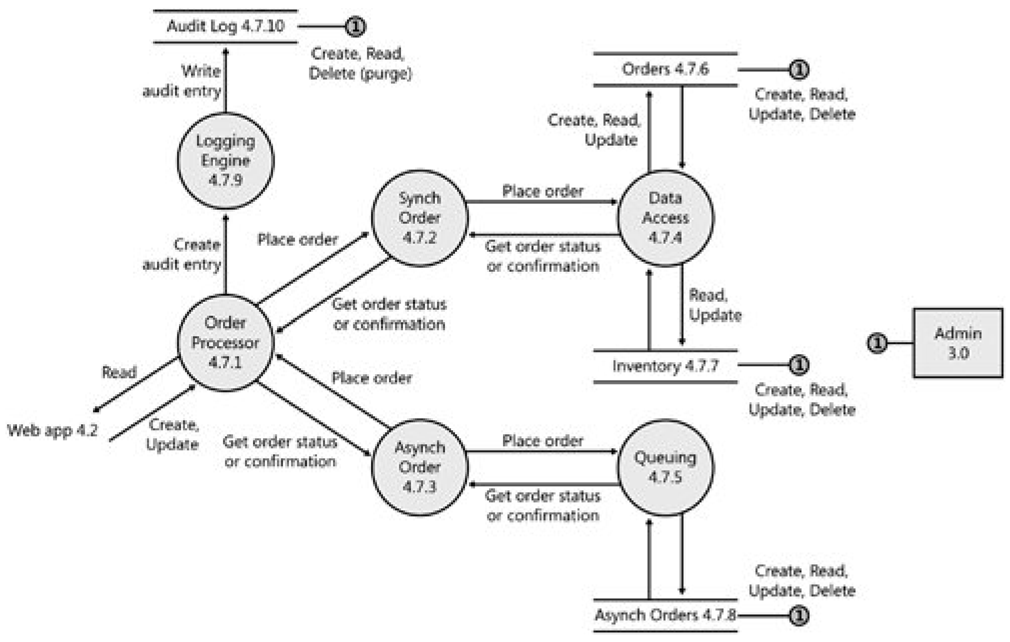
[View full size image]



As you can see, it's a little more complex when you drill down. You'll also notice one more complex process in the level-0 DFD: the order-processing process (4.7). We need to drill down into that, too, until there are no complex processes left. In the interests of clarity, we reduce the number of data flows from the administrator to the various data stores she manages. Figure 9-5 shows the level-1 DFD for order processing.

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

[View full size image]



At this level, we can see that the Pet Shop 4.0 application handles both synchronous and asynchronous requests. Asynchronous orders generally allow for greater application throughput. In this application, throughput increases by up to 30 percent. In the Pet Shop 4.0 application, asynchronous orders are not fulfilled immediately; rather, they are fulfilled by an external application that is not covered by this threat model.

Note that the numbering system for each of the DFD elements is a hierarchical dotted notation. Element 4.0 in the original context diagram (Figure 9-3) contains elements 4.1 through 4.9 in the level-0 DFD, and element 4.7 contains elements 4.7.1 through 4.7.10.

Important

The original Pet Shop 4.0 application did not include an audit-log facility; we added it (4.7.9 and 4.7.10) and made a design-change request to the Pet Shop authors. Programs such as online e-commerce applications *need* auditing facilities beyond those offered by the Web server and the database engines because many of the ordering semantics are lost to the Web and database servers that see only HTTP requests and database queries.

6. Determine Threat Types

Microsoft uses a threat taxonomy called STRIDE to identify various threat types. STRIDE considers threats from the attacker's perspective. Another common taxonomy is CIA, described below, which defines desirable security properties. STRIDE is more complete than CIA in that CIA does not address common security issues today, such as authentication. The properties of CIA are as follows:

- **Confidentiality** Ensures that information is not accessed by unauthorized users
- **Integrity** Ensures that information is not altered by unauthorized users in a way that is

undetectable by authorized users

- **Availability** Ensures that principals (users or computers) have appropriate access to resources

The following sections describe the components of STRIDE.

Spoofing Identity

Spoofing threats allow an attacker to pose as something or somebody else, such as another user pretending to be Bill Gates, a server pretending to be Microsoft.com, or even code posing as Ntdll.dll.

Tampering

Tampering threats involve malicious modification of data or code. The data or code being manipulated could be at rest or ephemeral "on-the-wire" data.

Repudiation

An attacker makes a repudiation threat by denying to have performed an action that other parties can neither confirm nor contradict. For example, a user makes a repudiation threat when he performs an illegal operation in a system that can't trace the prohibited operation.

Non-repudiation is a system's ability to counter repudiation threats. For example, in a commercial system, if a user signs for a purchased item upon receipt, the vendor can later use the signed receipt as evidence that the user received the item. As you can imagine, non-repudiation is important for e-commerce applications.

An important caveat you should understand about repudiation is that only humans repudiate (Ellison 2000). However, computers and software can gather evidence that can be used to counter the claims of the repudiating party.

Information Disclosure

Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it. Examples of this type of threat include a user's ability to read a file that she was not granted access to and an intruder's ability to read data in transit between computers.

Caution

Information disclosure threats, if left unmitigated, can become privacy violations if the disclosed data is confidential or personally identifiable information (PII).

Denial of Service

Denial-of-service (DoS) attacks deny or degrade service to valid usersfor example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.

Elevation of Privilege

Elevation-of-privilege (EoP) threats often occur when a user gains increased capability, often as an anonymous user who takes advantage of a coding bug to gain admin or root capability. One of the many examples of this threat is a series of defects in the open-source version-control softwareConcurrent Versions System, or CVSon Linux (CERT 2003), which led to a compromise of the kernel source code (Silicon 2003).

EoP is more subtle than as previously described because it does not apply only to anonymous or low-trust users. For example, the ability to go from an anonymous to a valid user is an EoP, as is moving from user to admin.

EoP threats also exist in code; for example, if a flaw in the Java or .NET runtime grants a unit of code more permission than normal, that too is an EoP threat. Java Web Start Untrusted Application Privilege Escalation (CVE-2005-1974) is an example of this threat. Likewise, running mobile code in a Web browser with more capability than the code should have is an EoP. Firefox/Mozilla Chrome UI DOM Property Override Privilege Escalation (CVE-2005-1160) is an example of this kind of EoP, as is the URL Decoding Zone Spoofing Vulnerability in Internet Explorer (CVE-2005-0054).

7. Identify Threats to the System

Once the DFD is done, you need to list all the DFD elements (also often referred to as assets) because you need to protect these elements from attack. Table 9-3 lists all the elements in the preceding DFD diagrams. Note that you don't include complex processes; rather, you include the processes, data stores, and data flows inside the complex process. However, you do model data flows in and out of a complex process.

Table 9-3. DFD Elements Within the Pet Shop 4.0 Application

DFD Element Type	DFD Item Numbers
External Entities	Pet Shop customer (1.0)
	Anonymous user (2.0)
	Administrator (3.0)
Processes	Web application (4.2)
	User profile (4.5)
	Membership (4.6)
	Order processor (4.7.1)
	Synchronous order processor (4.7.2)
	Asynchronous order processor (4.7.3)
	Data access component (4.7.4)

	Queuing component (4.7.5)
	Auditing engine (4.7.9)
Data Stores	Web application configuration data (4.1)
	Web pages (4.3)
	User profile data (4.8)
	Membership data (4.9)
	Orders data (4.7.6)
	Inventory data (4.7.7)
	Asynch orders data (4.7.8)
	Audit-log data (4.7.10)
Data Flows (partial list for brevity)	Anonymous user request (2.0→4.2)
	Anonymous user response (4.2→2.0)
	Pet Shop customer request (1.0→4.2)
	Pet Shop customer response (4.2→1.0)
	Web application reading configuration data (4.1→4.2)
	Web pages read by Web application (4.3→4.2)
	Admin creating or updating Web application configuration data (3.0→4.1)
	Admin reading Web application configuration data (4.1 →3.0)
	Admin creating, updating, or deleting Web pages (3.0→4.3)
	Admin reading Web pages (4.3→3.0)
	Web application creating or updating an order (4.2→4.7.1)
	Web application reading an order (4.7.1→4.2)

Now we have a list of all the DFD elements or assets within the application. Note that nearly all of the data flows are bidirectional. You should think of them as being separate for the time being.

You can apply a process called *reduction* to reduce the number of entities you will analyze. In short, if you have two or more DFD elements of the same type (for example, two or more processes) behind the same trust boundary, you can model the elements as one entity, as long as the elements were written in or are using the same technology and are handling similar data. In other words, when you analyze the threats to one of the elements, that same analysis applies to the other element also. You can see the biggest benefit of this process when you reduce data flows, which tend to be numerous. The first consideration is whether to reduce the bidirectional flows. For example, the anonymous user request and response (2.0⟶4.2 and 4.2⟶2.0) can be collapsed because:

- The data flows use the same technology (HTTP over TCP).
- They share the same process and external entity (2.0 and 4.2).
- The data content in either direction is public and anonymous.

The same reduction process applies to a Pet Shop customer request and response (1.0⟶4.2 and 4.2⟶1.0). The only difference is that the data flows carry authentication information, too, and the user can place orders and look up order information. You can also reduce the Web pages used for configuration (4.3⟶3.0 and 3.0⟶4.3), Web application configuration data (4.1⟶3.0 and 3.0⟶4.1), and Web application reading or manipulating order information (4.2⟶4.7.1 and 4.7.1⟶4.2).

The synchronous and asynchronous order-processing processes, 4.7.2 and 4.7.3, can also be reduced because they handle exactly the same data types, are within the same trust boundary, and are written in the same language (C#). The same applies to the data-access (4.7.4) and queuing (4.7.5) components and the orders (4.7.6) and asynchronous orders (4.7.8) data stores.

So now, after reduction, the abbreviated list of DFD elements looks like that in Table 9-4.

Table 9-4. Reduced DFD Elements Within Pet Shop 4.0

DFD Element Type	DFD Item Number
External Entities	Pet Shop customer (1.0)
	Anonymous user (2.0)
	Administrator (3.0)
Processes	Web application (4.2)
	User profile (4.5)
	Membership (4.6)
	Order processor (4.7.1)
	Sync/Async order processors (4.7.2 and 4.7.3)
	Data-access or queuing components (4.7.4 and 4.7.5)
	Auditing engine (4.7.9)
Data Stores	Web application configuration data (4.1)
	Web pages (4.3)

	Use profile data (4.8)
	Membership data (4.9)
	Order and async orders data (4.7.6 and 4.7.8)
	Inventory data (4.7.7)
	Audit-log data (4.7.10)
Data Flows (partial list)	Web application reading configuration data (4.1⟶4.2)
	Web pages read by Web application (4.3⟶4.2)
	Anonymous user request/response (2.0⟶4.2⟶2.0)
	Pet Shop customer request/response (1.0⟶4.2⟶1.0)
	Admin reading, creating, updating Web application configuration data (3.0⟶4.1⟶3.0)
	Admin reading, creating, updating, deleting Web pages (3.0⟶4.3⟶3.0)
	Web application reading, creating, updating an order (4.2 ⟶4.7.1⟶4.2)

Note

We have already defined STRIDE, but to save you time, here's what the acronym means: spoofing, tampering, repudiation, information disclosure, DoS, and EoP. Remember, STRIDE looks at threats from an attacker's perspective.

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 in Table 9-5.

Table 9-5. Mapping STRIDE to DFD Element Types

DFD Element Type	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	†	X	X	
Process	X	X	X	X	X	X

In essence, everything in the DFD is subject to attack, and the nature of the potential attack is determined by the DFD element type. For example, a data flow is subject to tampering, information disclosure, and DoS attacks.

Note

Most, but not all, DoS attacks against data stores and data flows are against a process at one end of the data flow or the process serving the data store.

Note the dagger mark (†) at the intersection of the data store row and the repudiation column. If the data store contains logging or audit data, repudiation is a potential threat because if the data is maliciously manipulated in any way, an attacker could cover his or her tracks, or a criminal could renege on a transaction. Of all the data stores in the DFDs, only one, the audit data store (4.7.10), is of concern from a repudiation perspective. The audit data store is important because it holds process-ordering audit information such as the following:

- Transaction date and time
- Pet Shop customer ID
- Pet Shop customer IP address
- Order transaction ID

The order transaction ID can then be used to cross-reference with the actual order in the ordering system.

The term *spoofing* is often misused in the context of security; many spoofing threats are in fact tampering threats. Replacing a file with a bogus file or changing bytes in a file is tampering. A real example of process spoofing would be spoofing a Web server. Ordinarily, the attacker does not have direct access to the Web server software serving Web pages for say, Microsoft.com, but the attacker might be able to spoof the site, perhaps using cache or Domain Name System (DNS) poisoning. However, one could argue that these attacks are really tampering threats against a data store (the user's cache, a proxy's cache, or the DNS server records.) The security of the appropriate caches and servers could be treated as a security assumptionyou are assuming that all the appropriate caches and infrastructure servers (DNS, DHCP, and so on) are performing correctly.

A variant of spoofing is *typosquatting*, in which an attacker creates a valid domain name, such as Micros0ft.com (note the zero instead of the letter *o*) and then builds a Web page that looks like valid Microsoft content. This is a very low-tech attack.

Important

Sometimes a perceived threat might in fact be a security assumption violation. For example, a spoofed Web site could manifest itself as a corrupted or tampered-with hosts file on a user's computer.

In Table 9-6, we combine the list of DFD elements with the STRIDE mappings to arrive at a list of threats to the system being modeled. To do this, we gather the elements from Table 9-4 and then determine the threats to which each is susceptible by using Table 9-5.

Table 9-6. Determining Threats for DFD Elements Within Pet Shop 4.0

DFD Element Type	Threat Types (STRIDE)	DFD Item Numbers
External entities	SR	(1.0), (2.0), (3.0)
Processes	STRIDE	(4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9)
Data stores	T(R)ID	(4.1), (4.3), (4.8), (4.9), (4.7.6 and 4.7.8), (4.7.7), (4.7.10 repudiation)
Data flows (partial list for brevity)	TID	(4.1⟶4.2), (4.3⟶4.2), (2.0⟶4.2⟶2.0), (1.0⟶4.2⟶1.0), (3.0⟶4.1⟶3.0), (3.0⟶4.3⟶3.0), (4.2⟶4.7.1⟶4.2)

Notice the parentheses around the letter *R* in the data stores row; a data store might be subject to repudiation threats if the data held in the store is logging or auditing data. In our Pet Shop example, data store 4.7.10 is subject to repudiation threats because it stores auditing data.

Table 9-7 gives us a complete list of potential threats to the system by showing the information from Table 9-6 a different way.

Table 9-7. Threats to the System

Threat Type (STRIDE)	DFD Item Numbers
Spoofing	External entities: (1.0), (2.0), (3.0) Processes: (4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9)
Tampering	Processes: (4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9) Data stores: (4.1), (4.3), (4.8), (4.9), (4.7.6 and 4.7.8), (4.7.7), (4.7.10) Data flows: (4.1⟶4.2), (4.3⟶4.2), (2.0⟶4.2⟶2.0), (1.0⟶4.2⟶1.0), (3.0⟶4.1⟶3.0), (3.0⟶4.3⟶3.0), (4.2⟶4.7.1⟶4.2)
Repudiation	External entities: (1.0), (2.0), (3.0) Data flow: (4.7.10)
Information disclosure	Processes: (4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9) Data stores: (4.1), (4.3), (4.8), (4.9), (4.7.6 and 4.7.8), (4.7.7), (4.7.10)

	Data flows: (4.1→4.2), (4.3→4.2), (2.0→4.2→2.0), (1.0→4.2→1.0), (3.0→4.1→3.0), (3.0→4.3→3.0), (4.2→4.7.1→4.2)
DoS	Processes: (4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9) Data stores: (4.1), (4.3), (4.8), (4.9), (4.7.6 and 4.7.8), (4.7.7), (4.7.10) Data flows: (4.1→4.2), (4.3→4.2), (2.0→4.2→2.0), (1.0→4.2→1.0), (3.0→4.1→3.0), (3.0→4.3→3.0), (4.2→4.7.1→4.2)
EoP	Processes: (4.2), (4.5), (4.6), (4.7.1), (4.7.2 and 4.7.3), (4.7.4 and 4.7.5), (4.7.9)

Now that we have this list, it's time to look at the potential risk of each threat.

8. Determine Risk

Historically, security specialists have used numeric calculations to determine risk. The problem with using numbers is they can be very subjective. For example, Microsoft often used DREAD ratings (Damage potential, Reproducibility, Exploitability, Affected users, Discoverability) to calculate risk, and sometimes people used a calculation like this one:

Risk = Chance of Attack x *Damage Potential*

Simply put, the problem is determining the chance of attack. You can't predict the future, so you have no idea, other than a guess, what the chance of attack really is. We're not saying security risk calculation using numbers is uselessit's notbut it's very hard to be consistent and accurate (especially as team members move around), so use numeric calculations with caution.

Microsoft has created a *bug bar* that defines the characteristics of a threat and, thereby, the level of risk. Rather than use numbers, the risk rankings are derived in part from the Microsoft Security Response Center (MSRC) security bulletin rankings. We will use the term "risk level" to indicate overall risk; risk level 1 is highest and risk level 4 is the lowest. The characteristics of a threat include:

- Server application (for example, an e-mail server) versus client application (for example, a word processor).
- Local versus remote accessibility.
- Accessibility to anonymous versus authenticated users.
- Accessibility to authenticated users versus administrators.
- On by default versus off by default.
- The degree of user interaction required.

- In the case of an information disclosure threat, whether the data is personally identifiable information (PII) or is sensitive data.
- In the case of a DoS attack, whether the application continues service or is nonfunctional once an attack stops.

Figures 9-6 through 9-10 illustrate an abbreviated set of trees outlining the core elements of the Security Development Lifecycle (SDL) bug bar document.

Figure 9-6. Spoofing threats risk ranking.

[View full size image]

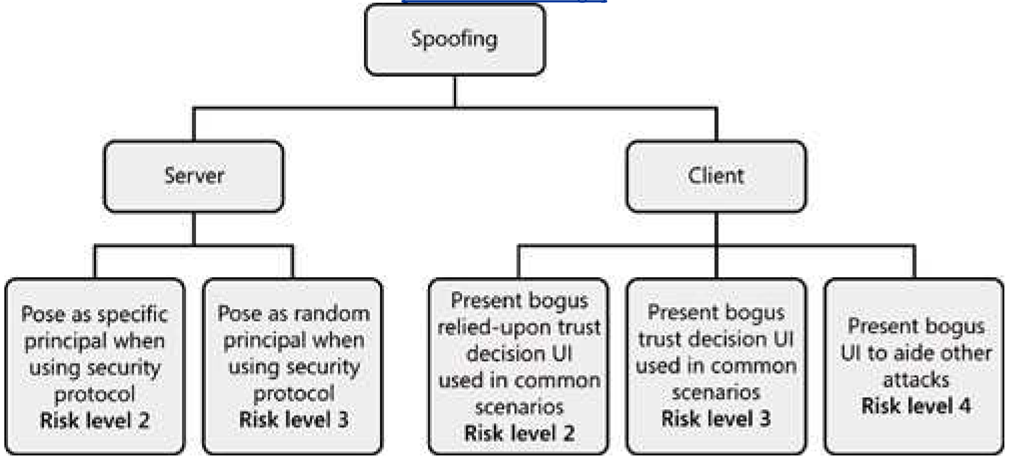
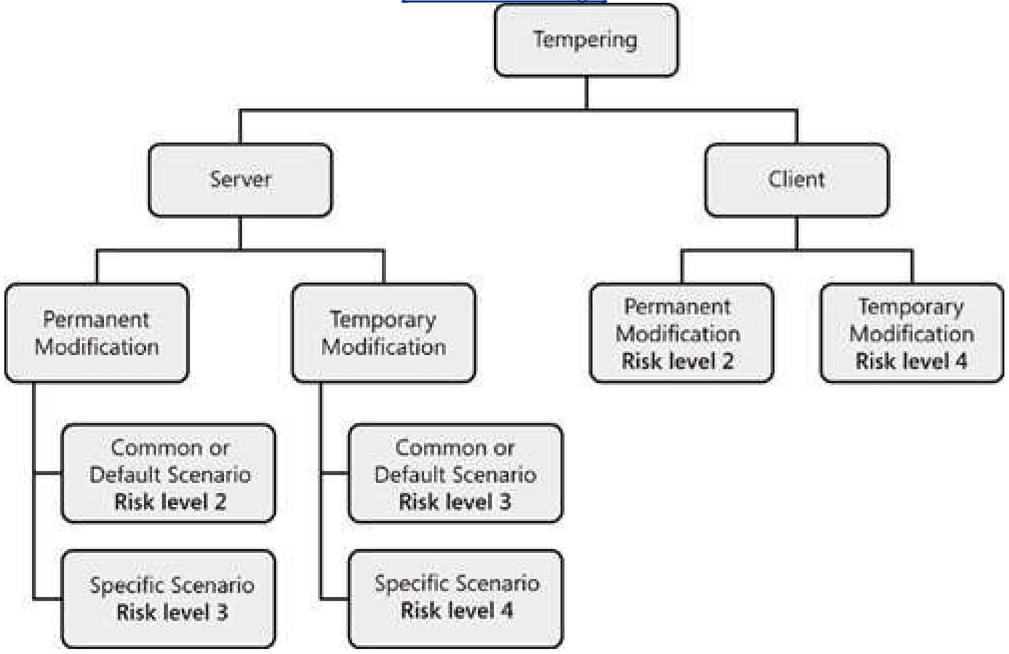

Spoofing
Server
Client
Pose as specific principal when using security protocol Risk level 2
Pose as random principal when using security protocol Risk level 3
Present bogus relied-upon trust decision UI used in common scenarios Risk level 2
Present bogus trust decision UI used in common scenarios Risk level 3
Present bogus UI to aide other attacks Risk level 4

Figure 9-7. Tampering threats risk ranking.

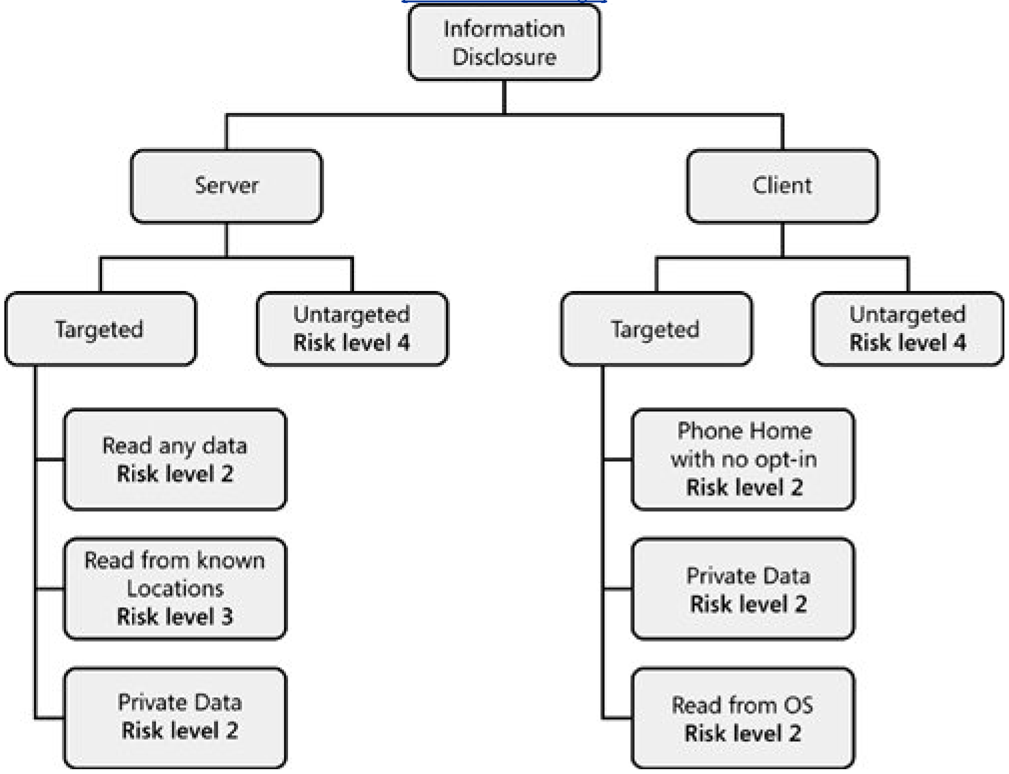
[View full size image]


Tampering
Server
Client
Permanent Modification
Temporary Modification
Permanent Modification Risk level 2
Temporary Modification Risk level 4
Common or Default Scenario Risk level 2
Specific Scenario Risk level 3
Common or Default Scenario Risk level 3
Specific Scenario Risk level 4

In Figure 9-8, *target* means the ability to disclose selected data. An example of an untargeted attack is one that perhaps yields random heap data.

Figure 9-8. Information disclosure threats risk ranking.

[View full size image]


Information Disclosure
Server
Client
Targeted
Untargeted Risk level 4
Targeted
Untargeted Risk level 4
Read any data Risk level 2
Read from known Locations Risk level 3
Private Data Risk level 2
Phone Home with no opt-in Risk level 2
Private Data Risk level 2
Read from OS Risk level 2

In Figure 9-9, *with amplification* means the attack leads to an increase in denied service. For example, an attacked computer begins to attack all computers on the subnet, as in the TCP/IP *smurf* attack.

Figure 9-9. DoS threats risk ranking.

[View full size image]

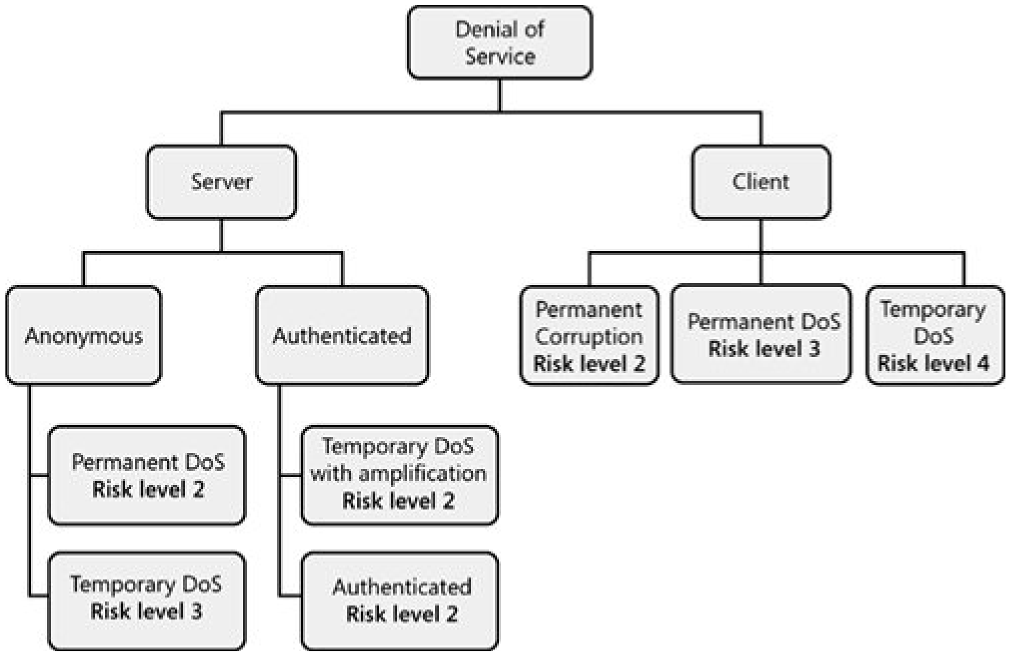
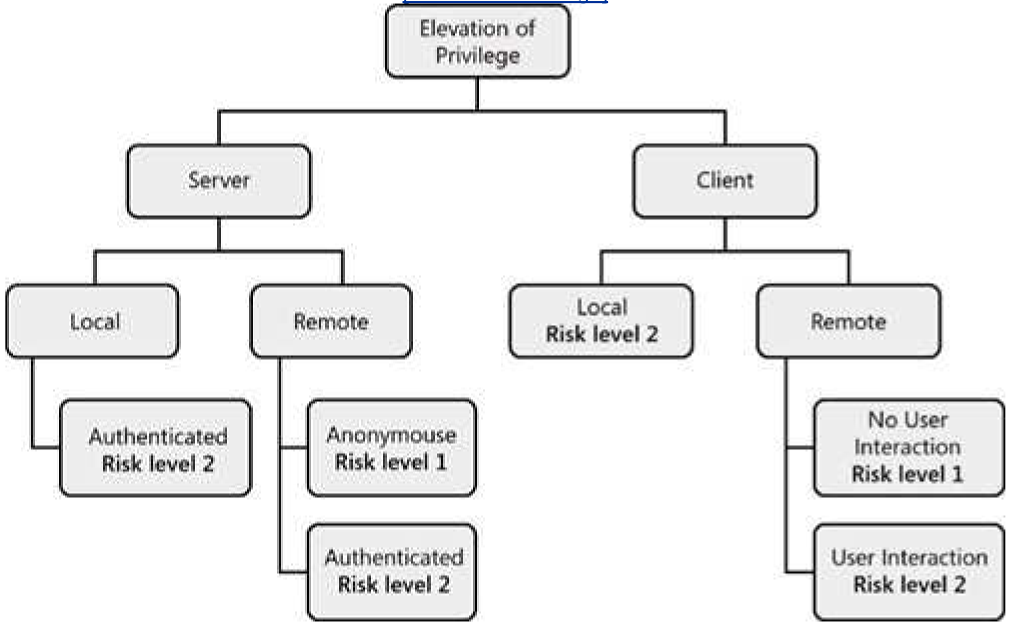

Denial of Service
Server
Client
Anonymous
Authenticated
Permanent Corruption Risk level 2
Permanent DoS Risk level 3
Temporary DoS Risk level 4
Permanent DoS Risk level 2
Temporary DoS Risk level 3
Temporary DoS with amplification Risk level 2
Authenticated Risk level 2

Figure 9-10. EoP threats risk ranking.

[View full size image]


Elevation of Privilege
Server
Client
Local
Remote
Local Risk level 2
Remote
Authenticated Risk level 2
Anonymouse Risk level 1
Authenticated Risk level 2
No User Interaction Risk level 1
User Interaction Risk level 2

Note that there are no risk rankings for repudiation threats. The threats remain unranked for various reasonsmost notably because Microsoft has issued no security bulletins relating to repudiation. In general, the feeling is that any such errors follow the tampering risk rankings.

By the end of the risk-identification stage, you should have ranked your threats by risk, from high to low. Obviously, you should address the highest-risk items first. Risk level 1 or 2 threats must always be remedied during the development phase. Risk level 3 threats should be fixed before the product becomes a release candidate, and risk level 4 threats should be fixed if time permits.

9. Plan Mitigations

Often referred to as countermeasures or defenses, mitigations reduce or eliminate the risk of a threat. You have only a small set of mitigation strategies available to you:

- Do nothing.
- Remove the feature.
- Turn off the feature.
- Warn the user.
- Counter the threat with technology.

Let's look at each option in detail.

Do Nothing

For low-risk threats, doing nothing could be a valid strategy. However, you might find it worthwhile to update your external-security notes to reflect such threats.

Remove the Feature

Removing a feature is the only way to reduce the risk to zero. We have done this numerous times at Microsoft when threat models indicate that the risk is too great or the mitigations are untenable. In those cases, the best course of action is to not build the feature in the first place. Obviously, this is a balancing act between user features and potential security risks.

Turn Off the Feature

Turning off a feature is less drastic than removing a feature and should be used only to reduce risk further. Building a threat-ridden feature and then simply turning it off by default is unacceptable. Consider this only as a defense-in-depth strategy.

Warn the User

For some threats, you should warn the user. However, be aware that most users, especially non-technical users, make poor trust decisions. Beware of threat models that are rife with user-warning mitigations.

Counter the Threat with Technology

Countering threats with technology is the most common mitigation strategy. Use a technology such as authentication or encryption to solve specific issues. To determine what mitigations to use, consider threat types. Table 9-8 outlines high-level mitigation strategies by threat type.

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

For each of the mitigation techniques, you can use one or more technologies. Table 9-9 gives some examples.

Table 9-9. Mitigation Technologies

Mitigation Technique	Mitigation Technology
Authentication	Authenticate principals: • Basic authentication • Digest authentication • Cookie authentication • Windows authentication (NTLM) • Kerberos authentication • PKI systems such as SSL/TLS and certificates • IPSec • Digitally signed packets Authenticate code or data: • Digital signatures • Message authentication codes • Hashes
Integrity	• Windows Vista Mandatory Integrity Controls • ACLs • Digital signatures

	• Message authentication codes
Non-repudiation services	• Strong authentication • Secure auditing and logging • Digital signatures • Secure time-stamps • Trusted third parties
Confidentiality	• ACLs • Encryption
Availability	• ACLs • Filtering • Quota • Authorization
Authorization	• ACLs • Group or role membership • Privilege ownership • Permissions

A complete list of mitigation mechanisms and best practices for choosing the appropriate mitigation is beyond the scope of this book, but much literature on the topic exists (Viega and McGraw 2001, Ferguson and Schneier 2003, Howard and LeBlanc 2003).

The next step in countering threats is to take all the threats from the threat model, look up the mitigation techniques, and then determine an appropriate mitigation technology.

Table 9-10 provides a list of some "interesting" assets from the Pet Shop 4.0 example. The list is highly abbreviated and is intended only to give an example of the final stages of the threat-modeling process.

Table 9-10. Abbreviated List of Interesting Threats to Pet Shop 4.0

Example Asset	Asset Type	Threat Type Susceptibility	Example Threat
(1.0→4.2	Data flow from Pet Shop user	TID	I

⟶1.0)	to Web application and back		
4.7.10	Audit log data store	T(R)ID	T
4.7.1	Order processor process	TRIDE	STRIDE

Now let's look at the potential mitigations.

(1.0⟶4.2⟶1.0) Data Flow from Pet Shop User to Web Application and Back

Data flows are subject to tampering, information disclosure, and DoS attacks. We'll focus on the information disclosure threat because the data flow from the Pet Shop user to the Web application and back could contain potentially sensitive data such as the user name and password and credit card information. The mitigation technique for such threats is confidentiality, and in this case, we can use SSL/TLS because

- High-level protocols are always a preferred way to mitigate threats because they are well proven and easy to implement.
- SSL/TLS solves the confidentiality problem through encryption.
- SSL/TLS solves the tampering threat by using Message Authentication Codes (MACs).
- SSL/TLS solves the spoofing threat to the Web application (4.2) by providing authentication services.

(4.7.10) Audit Log Data Store

Data stores are subject to tampering, information disclosure, DoS, and, potentially, repudiation attacks. In this audit log example, we're going to look at the tampering threat because this could lead to repudiation issues. If an attacker can tamper with the data log store, he can cover his tracks or repudiate his transactions.

Tampering threats are mitigated with integrity technologies such as ACLs, hashes, digital signatures, or MACs. You should use a good ACL on the log file that allows only the logging process and trusted users to manipulate the file. Then you should use a signature or message authentication code on the file as well, depending on your business requirements.

(4.7.1) Order Processor Process

Finally, processes are susceptible to spoofing, tampering, information disclosure, DoS, and EoP attacks. The tampering threat for the processor process can be mitigated in the same way the tampering threat is mitigated for the audit log file: use an ACL and a MAC or digital signature. In the case of a process, you'll probably use a signature; these are commonly used to protect code. The two mitigations for the EoP threat are first to authorize only valid users to access the code, and second to always run the code by using the lowest-possible privilege.

We finally have a list of mitigations we can build into the design of the Pet Shop 4.0 application. Table 9-11 summarizes our defenses.

Table 9-11. Defenses Used in Portions of Pet Shop 4.0

Asset	Asset Type	Example Threat	Example Mitigation
(1.0⟶4.2 ⟶1.0)	Data flow from Pet Shop user to Web application and back	I	SSL/TLS (also mitigates the tampering threat)
4.7.10	Audit log data store	T	ACL and MAC
4.7.1	Order processor process	T and E	ACL, MAC, and reduced process privilege

Role of Threat Trees

If you are familiar with other texts that cover threat modeling, you may notice that threat trees are not covered in this chapter. This is by design. Threat trees, although very useful, require a lot of security expertise to build and use correctly. One of the most frequent criticisms we face about the earlier threat-modeling process is the level of security expertise required to build a good threat model. Analysis revealed that the key weakness was building the threat trees. So rather than expecting nonsecurity experts to build accurate and consistent trees, we have removed them and replaced them with threat tree patterns. These trees illustrate common attack patterns and allow the application designers to think about other conditions in the system. You'll find a complete list of threat trees in Chapter 22, "Threat Tree Patterns."

After you build your threat model, you can consult the trees to determine other conditions to consider. For example, if you have a spoofing threat against a user, you can consult the "Spoofing Threats Against External Entities and Processes" tree and look at the leaf nodes. These will prompt you to ask other relevant questions about the design of the system.

PREV NEXT