CHAPTER 3

STRIDE

As you learned in Chapter 1, "Dive in and Threat Model!," STRIDE is an acronym that stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. The STRIDE approach to threat modeling was invented by Loren Kohnfelder and Praerit Garg (Kohnfelder, 1999). This framework and mnemonic was designed to help people developing software identify the types of attacks that software tends to experience.

The method or methods you use to think through threats have many different labels: finding threats, threat enumeration, threat analysis, threat elicitation, threat discovery. Each connotes a slightly different flavor of approach. Do the threats exist in the software or the diagram? Then you're finding them. Do they exist in the minds of the people doing the analysis? Then you're doing analysis or elicitation. No single description stands out as always or clearly preferable, but this book generally talks about finding threats as a superset of all these ideas. Using STRIDE is more like an elicitation technique, with an expectation that you or your team understand the framework and know how to use it. If you're not familiar with STRIDE, the extensive tables and examples are designed to teach you how to use it to discover threats.

This chapter explains what STRIDE is and why it's useful, including sections covering each component of the STRIDE mnemonic. Each threat-specific section provides a deeper explanation of the threat, a detailed table of examples for that threat, and then a discussion of the examples. The tables and examples are designed to teach you how to use STRIDE to discover threats. You'll also learn about approaches built on STRIDE: STRIDE-per-element, STRIDE-per-interaction, and DESIST. The other approach built on STRIDE, the *Elevation of Privilege* game, is covered in Chapters 1, "Dive In and Threat Model!" and 12, "Requirements Cookbook," and Appendix C, "Attacker Lists."

Understanding STRIDE and Why It's Useful

The STRIDE threats are the opposite of some of the properties you would like your system to have: authenticity, integrity, non-repudiation, confidentiality, availability, and authorization. Table 3-1 shows the STRIDE threats, the corresponding property that you'd like to maintain, a definition, the most typical victims, and examples.

Table 3-1: The STRIDE Threats

THREAT	PROPERTY VIOLATED	THREAT DEFINITION	TYPICAL VICTIMS	EXAMPLES
Spoofing	Authentication	Pretending to be something or someone other than yourself	Processes, external entities, people	Falsely claiming to be Acme.com, winsock .dll, Barack Obama, a police officer, or the Nigerian Anti-Fraud Group
Tampering	Integrity	Modifying something on disk, on a network, or in memory	Data stores, data flows, processes	Changing a spreadsheet, the binary of an important program, or the contents of a database on disk; modifying, adding, or removing packets over a network, either local or far across the Internet, wired or wireless; changing either the data a program is using or the running program itself

THREAT	PROPERTY VIOLATED	THREAT DEFINITION	TYPICAL VICTIMS	EXAMPLES
Repudiation	Non-Repudiation	Claiming that you didn't do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?	Process	Process or system: "I didn't hit the big red button" or "I didn't order that Ferrari." Note that repudiation is somewhat the odd-threat-out here; it transcends the technical nature of the other threats to the business layer.
Information Disclosure	Confidentiality	Providing information to someone not authorized to see it	Processes, data stores, data flows	The most obvious example is allowing access to files, e-mail, or databases, but information disclosure can also involve filenames ("Termination for John Doe.docx"), packets on a network, or the contents of program memory.
Denial of Service	Availability	Absorbing resources needed to provide service	Processes, data stores, data flows	A program that can be tricked into using up all its memory, a file that fills up the disk, or so many network connections that real traffic can't get through
Elevation of Privilege	Authorization	Allowing someone to do something they're not authorized to do	Process	Allowing a normal user to execute code as admin; allowing a remote person without any privileges to run code

In Table 3-1, "typical victims" are those most likely to be victimized: For example, you can spoof a program by starting a program of the same name, or by putting a program with that name on disk. You can spoof an endpoint on the same machine by squatting or splicing. You can spoof users by capturing their authentication info by spoofing a site, by assuming they reuse credentials across sites, by brute forcing (online or off) or by elevating privilege on their machine. You can also tamper with the authentication database and then spoof with falsified credentials.

Note that as you're using STRIDE to look for threats, you're simply enumerating the things that might go wrong. The exact mechanisms for how it can go wrong are something you can develop later. (In practice, this can be easy or it can be very challenging. There might be defenses in place, and if you say, for example, "Someone could modify the management tables," someone else can say, "No, they can't because...") It can be useful to record those possible attacks, because even if there is a mitigation in place, that mitigation is a testable feature, and you should ensure that you have a test case.

You'll sometimes hear STRIDE referred to as "STRIDE categories" or "the STRIDE taxonomy." This framing is not helpful because STRIDE was not intended as, nor is it generally useful for, categorization. It is easy to find things that are hard to categorize with STRIDE. For example, earlier you learned about tampering with the authentication database and then spoofing. Should you record that as a tampering threat or a spoofing threat? The simple answer is that it doesn't matter. If you've already come up with the attack, why bother putting it in a category? The goal of STRIDE is to help you find attacks. Categorizing them might help you figure out the right defenses, or it may be a waste of effort. Trying to use STRIDE to categorize threats can be frustrating, and those efforts cause some people to dismiss STRIDE, but this is a bit like throwing out the baby with the bathwater.

Spoofing Threats

Spoofing is pretending to be something or someone other than yourself. Table 3-1 includes the examples of claiming to be Acme.com, winsock.dll, Barack Obama, or the Nigerian Anti-Fraud Office. Each of these is an example of a different subcategory of spoofing. The first example, pretending to be Acme.com (or Google.com, etc.) entails spoofing the identity of an entity across a network. There is no mediating authority that takes responsibility for telling you that Acme.com is the site I mean when I write these words. This differs from the second example, as Windows includes a `winsock.dll`. You should be able to ask the operating system to act as a mediating authority and get you to `winsock`. If you have your own DLLs, then you need to ensure that you're opening them with the appropriate path (`%installdir%\dll`); otherwise, someone might substitute one in a working directory, and get your code to do what they want. (Similar issues exist with unix and `LD_PATH`.) The third example, spoofing Barack Obama, is an instance of pretending to be a specific person. Contrast that with the fourth example, pretending to be the President of the United States or the Nigerian Anti-Fraud Office. In those cases, the attacker is pretending to be in a role. These spoofing threats are laid out in Table 3-2.

Table 3-2: Spoofing Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Spoofing a process on the same machine	Creates a file before the real process	
	Renaming/linking	Creating a Trojan "su" and altering the path
	Renaming	Naming your process "sshd"
Spoofing a file	Creates a file in the local directory	This can be a library, executable, or config file.
	Creates a link and changes it	From the attacker's perspective, the change should happen between the link being checked and the link being accessed.
	Creates many files in the expected directory	Automation makes it easy to create 10,000 files in `/tmp`, to fill the space of files called `/tmp /"pid.NNNN,` or similar.
Spoofing a machine	ARP spoofing	
	IP spoofing	
	DNS spoofing	Forward or reverse
	DNS Compromise	Compromise TLD, registrar or DNS operator
	IP redirection	At the switch or router level
Spoofing a person	Sets e-mail display name	
	Takes over a real account	
Spoofing a role	Declares themselves to be that role	Sometimes opening a special account with a relevant name

Spoofing a Process or File on the Same Machine

If an attacker creates a file before the real process, then if your code is not careful to create a new file, the attacker may supply data that your code interprets, thinking that your code (or a previous instantiation or thread) wrote that data, and it can be trusted. Similarly, if file permissions on a pipe, local procedure call, and so on, are not managed well, then an attacker can create that endpoint, confusing everything that attempts to use it later.

Spoofing a process or file on a remote machine can work either by creating spoofed files or processes on the expected machine (possibly having taken admin rights) or by pretending to be the expected machine, covered next.

Spoofing a Machine

Attackers can spoof remote machines at a variety of levels of the network stack. These spoofing attacks can influence your code's view of the world as a client, server, or peer. They can spoof ARP requests if they're local, they can spoof IP packets to make it appear that they're coming from somewhere they are not, and they can spoof DNS packets. DNS spoofing can happen when you do a forward or reverse lookup. An attacker can spoof a DNS reply to a forward query they expect you to make. They can also adjust DNS records for machines they control such that when your code does a reverse lookup (translating IP to FQDN) their DNS server returns a name in a domain that they do not control—for example, claiming that 10.1.2.3 is `update.microsoft.com`. Of course, once attackers have spoofed a machine, they can either spoof or act as a man-in-the-middle for the processes on that machine. Second-order variants of this threat involve stealing machine authenticators such as cryptographic keys and abusing them as part of a spoofing attack.

Attackers can also spoof at higher layers. For example, *phishing attacks* involve many acts of spoofing. There's usually spoofing of e-mail from "your" bank, and spoofing of that bank's website. When someone falls for that e-mail, clicks the link and visits the bank, they then enter their credentials, sending them to that spoofed website. The attacker then engages in one last act of spoofing: They log into your bank account and transfer your money to themselves or an accomplice. (It may be one attacker, or it may be a set of attackers, contracting with one another for services rendered.)

Spoofing a Person

Major categories of spoofing people include access to the person's account and pretending to be them through an alternate account. Phishing is a common way to get access to someone else's account. However, there's often little to prevent anyone from setting up an account and pretending to be you. For example, an attacker could set up accounts on sites like LinkedIn, Twitter, or Facebook and pretend to be you, the Adam Shostack who wrote this book, or a rich and deposed prince trying to get their money out of the country.

Tampering Threats

Tampering is modifying something, typically on disk, on a network, or in memory. This can include changing data in a spreadsheet (using either a program such as Excel or another editor), changing a binary or configuration file on disk, or modifying a more complex data structure, such as a database on disk. On a network, packets can be added, modified, or removed. It's sometimes easier to add packets than to edit them as they fly by, and programs are remarkably bad about handling extra copies of data securely. More examples of tampering are in Table 3-3.

Table 3-3: Tampering Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Tampering with a file	Modifies a file they own and on which you rely	
	Modifies a file you own	
	Modifies a file on a file server that you own	
	Modifies a file on their file server	Loads of fun when you include files from remote domains
	Modifies a file on their file server	Ever notice how much XML includes remote schemas?
	Modifies links or redirects	
Tampering with memory	Modifies your code	Hard to defend against once the attacker is running code as the same user
	Modifies data they've supplied to your API	Pass by value, not by reference when crossing a trust boundary
Tampering with a network	Redirects the flow of data to their machine	Often stage 1 of tampering
	Modifies data flowing over the network	Even easier and more fun when the network is wireless (WiFi, 3G, et cetera)
	Enhances spoofing attacks	

Tampering with a File

Attackers can modify files wherever they have write permission. When your code has to rely on files others can write, there's a possibility that the file was written maliciously. While the most obvious form of tampering is on a local disk, there are also plenty of ways to do this when the file is remotely included, like most of the JavaScript on the Internet. The attacker can breach your security by breaching someone else's site. They can also (because of poor privileges, spoofing, or elevation of privilege) modify files you own. Lastly, they can modify links or redirects of various sorts. Links are often left out of integrity checks. There's a somewhat subtle variant of this when there are caches between things you control (such as a server) and things you don't (such as a web browser on the other side of the Internet). For example, *cache poisoning attacks* insert data into web caches through poor security controls at caches (OWASP, 2009).

Tampering with Memory

Attackers can modify your code if they're running at the same privilege level. At that point, defense is tricky. If your API handles data by reference (a pattern often chosen for speed), then an attacker can modify it after you perform security checks.

Tampering with a Network

Network tampering often involves a variety of tricks to bring the data to the attacker's machine, where he forwards some data intact and some data modified. However, tricks to bring you the data are not always needed; with radio interfaces like WiFi and Bluetooth, more and more data flow through the air. Many network protocols were designed with the assumption you needed special hardware to create or read arbitrary packets. The requirement for special hardware was the defense against tampering (and often spoofing). The rise of software-defined radio (SDR) has silently invalidated the need for special hardware. It is now easy to buy an inexpensive SDR unit that can be programmed to tamper with wireless protocols.

Repudiation Threats

Repudiation is claiming you didn't do something, or were not responsible for what happened. People can repudiate honestly or deceptively. Given the increasing knowledge often needed to understand the complex world, those honestly repudiating may really be exposing issues in your user experiences or service architectures. Repudiation threats are a bit different from other security threats, as they often appear at the business layer. (That is, above the network layer such as TCP/IP, above the application layer such as HTTP/HTML, and where the business logic of buying products would be implemented.)

Repudiation threats are also associated with your logging system and process. If you don't have logs, don't retain logs, or can't analyze logs, repudiation threats are hard to dispute. There is also a class of attacks in which attackers will drop data in the logs to make log analysis tricky. For example, if you display your logs in HTML and the attacker sends `</tr>` or `</html>`, your log display needs to treat those as data, not code. More repudiation threats are shown in Table 3-4.

Table 3-4: Repudiation Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Repudiating an action	Claims to have not clicked	Maybe they really did
	Claims to have not received	Receipt can be strange; does mail being downloaded by your phone mean you've read it? Did a network proxy pre-fetch images? Did someone leave a package on the porch?
	Claims to have been a fraud victim	
	Uses someone else's account	
	Uses someone else's payment instrument without authorization	
Attacking the logs	Notices you have no logs	
	Puts attacks in the logs to confuse logs, log-reading code, or a person reading the logs	

Attacking the Logs

Again, if you don't have logs, don't retain logs, or can't analyze logs, repudiation actions are hard to dispute. So if you aren't logging, you probably need to start. If you have no log centralization or analysis capability, you probably need that as well. If you don't properly define what you will be logging, an attacker may be able to break your log analysis system. It can be challenging to work through the layers of log production and analysis to ensure reliability, but if you don't, it's easy to have attacks slip through the cracks or inconsistencies.

Repudiating an Action

When you're discussing repudiation, it's helpful to discuss "someone" rather than "an attacker." You want to do this because those who repudiate are often not actually attackers, but people who have been failed by technology or process. Maybe they really didn't click (or didn't perceive that they clicked). Maybe the spam filter really did eat that message. Maybe UPS didn't deliver, or maybe UPS delivered by leaving the package on a porch. Maybe someone claims to have been a victim of fraud when they really were not (or maybe someone else in a household used their credit card, with or without their knowledge). Good technological systems that both authenticate and log well can make it easier to handle repudiation issues.

Information Disclosure Threats

Information disclosure is about allowing people to see information they are not authorized to see. Some information disclosure threats are shown in Table 3-5.

Table 3-5: Information Disclosure Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Information disclosure against a process	Extracts secrets from error messages	
	Reads the error messages from username/passwords to entire database tables	
	Extracts machine secrets from error cases	Can make defense against memory corruption such as ASLR far less useful
	Extracts business/personal secrets from error cases	
Information disclosure against data stores	Takes advantage of inappropriate or missing ACLs	
	Takes advantage of bad database permissions	
	Finds files protected by obscurity	
	Finds crypto keys on disk (or in memory)	
	Sees interesting information in filenames	
	Reads files as they traverse the network	
	Gets data from logs or temp files	
	Gets data from swap or other temp storage	
	Extracts data by obtaining device, changing OS	

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Information disclosure against a data flow	Reads data on the network	
	Redirects traffic to enable reading data on the network	
	Learns secrets by analyzing traffic	
	Learns who's talking to whom by watching the DNS	
	Learns who's talking to whom by social network info disclosure	

Information Disclosure from a Process

Many instances in which a process will disclose information are those that inform further attacks. A process can do this by leaking memory addresses, extracting secrets from error messages, or extracting design details from error messages. Leaking memory addresses can help bypass ASLR and similar defenses. Leaking secrets might include database connection strings or passwords. Leaking design details might mean exposing anti-fraud rules like "your account is too new to order a diamond ring."

Information Disclosure from a Data Store

As data stores, well, store data, there's a profusion of ways they can leak it. The first set of causes are failures to properly use security mechanisms. Not setting permissions appropriately or hoping that no one will find an obscure file are common ways in which people fail to use security mechanisms. Cryptographic keys are a special case whereby information disclosure allows additional attacks. Files read from a data store over the network are often readable as they traverse the network.

An additional attack, often overlooked, is data in filenames. If you have a directory named "May 2013 layoffs," the filename itself, "Termination Letter for Alice.docx," reveals important information.

There's also a group of attacks whereby a program emits information into the operating environment. Logs, temp files, swap, or other places can contain data. Usually, the OS will protect data in swap, but for things like crypto keys, you should use OS facilities for preventing those from being swapped out.

Lastly, there is the class of attacks whereby data is extracted from the device using an operating system under the attacker's control. Most commonly (in 2013), these attacks affect USB keys, but they also apply to CDs, backup tapes, hard drives, or stolen laptops or servers. Hard drives are often decommissioned without full data deletion. (You can address the need to delete data from hard drives by buying a hard drive chipper or smashing machine, and since such machines are awesome, why on earth wouldn't you?)

Information Disclosure from a Data Flow

Data flows are particularly susceptible to information disclosure attacks when information is flowing over a network. However, data flows on a single machine can still be attacked, particularly when the machine is shared by cloud co-tenants or many mutually distrustful users of a compute server. Beyond the simple reading of data on the network, attackers might redirect traffic to themselves (often by spoofing some network control protocol) so they can see it when they're not on the normal path. It's also possible to obtain information even when the network traffic itself is encrypted. There are a variety of ways to learn secrets about who's talking to whom, including watching DNS, friend activity on a site such as LinkedIn, or other forms of social network analysis.

NOTE Security mavens may be wondering if side channel attacks and covert channels are going to be mentioned. These attacks can be fun to work on (and side channels are covered a bit in Chapter 16, "Threats to Cryptosystems"), but they are not relevant until you've mitigated the issues covered here.

Denial-of-Service Threats

Denial-of-service attacks absorb a resource that is needed to provide service. Examples are described in Table 3-6.

Table 3-6: Denial-of-Service Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Denial of service against a process	Absorbs memory (RAM or disk)	
	Absorbs CPU	
	Uses process as an amplifier	
Denial of service against a data store	Fills data store up	
	Makes enough requests to slow down the system	
Denial of service against a data flow	Consumes network resources	

Denial-of-service attacks can be split into those that work while the attacker is attacking (say, filling up bandwidth) and those that persist. Persistent attacks can remain in effect until a reboot (for example, `while(1){fork();}`), or even past a reboot (for example, filling up a disk). Denial-of-service attacks can also be divided into amplified and unamplified. Amplified attacks are those whereby small attacker effort results in a large impact. An example would take advantage of the old unix chargen service, whose purpose was to generate a semi-random character scheme for testing. An attacker could spoof a single packet from the chargen port on machine A to the chargen port on machine B. The hilarity continues until someone pulls a network cable.

Elevation of Privilege Threats

Elevation of privilege is allowing someone to do something they're not authorized to do—for example, allowing a normal user to execute code as admin, or allowing a remote person without any privileges to run code. Two important ways to elevate privileges involve corrupting a process and getting past authorization checks. Examples are shown in Table 3-7.

Table 3-7: Elevation of Privilege Threats

THREAT EXAMPLES	WHAT THE ATTACKER DOES	NOTES
Elevation of privilege against a process by corrupting the process	Send inputs that the code doesn't handle properly	These errors are very common, and are usually high impact.
	Gains access to read or write memory inappropriately	Writing memory is (hopefully obviously) bad, but reading memory can enable further attacks.
Elevation through missed authorization checks		
Elevation through buggy authorization checks		Centralizing such checks makes bugs easier to manage
Elevation through data tampering	Modifies bits on disk to do things other than what the authorized user intends	

Elevate Privileges by Corrupting a Process

Corrupting a process involves things like smashing the stack, exploiting data on the heap, and a whole variety of exploitation techniques. The impact of these techniques is that the attacker gains influence or control over a program's control flow. It's important to understand that these exploits are not limited to the attack surface. The first code that attacker data can reach is, of course, an important target. Generally, that code can only validate data against a limited subset of purposes. It's important to trace the data flows further to see where else elevation of privilege can take place. There's a somewhat unusual case whereby a program relies on and executes things from shared memory, which is a trivial path for elevation if everything with permissions to that shared memory is not running at the same privilege level.

Elevate Privileges through Authorization Failures

There is also a set of ways to elevate privileges through authorization failures. The simplest failure is to not check authorization on every path. More complex for an attacker is taking advantage of buggy authorization checks. Lastly, if a program relies on other programs, configuration files, or datasets being trustworthy, it's important to ensure that permissions are set so that each of those dependencies is properly secured.

Extended Example: STRIDE Threats against Acme-DB

This extended example discusses how STRIDE threats could manifest against the Acme/SQL database described in Chapter 1, "Dive In and Threat Model!" and 2, "Strategies for Threat Modeling," and shown in Figure 2-1. You'll first look at these threats by STRIDE category, and then examine the same set according to who can address them.

Spoofing

- A web client could attempt to log in with random credentials or stolen credentials, as could a SQL client.
- If you assume that the SQL client is the one you wrote and allow it to make security decisions, then a spoofed (or tampered with) client could bypass security checks.
- The web client could connect to a false (spoofed) front end, and end up disclosing credentials.
- A program could pretend to be the database or log analysis program, and try to read data from the various data stores.

Tampering

- Someone could also tamper with the data they're sending, or with any of the programs or data files.
- Someone could tamper with the web or SQL clients. (This is nominally out of scope, as you shouldn't be trusting external entities anyway.)

NOTE These threats, once you consider them, can easily be addressed with operating system permissions. More challenging is what can alter what data within the database. Operating system permissions will only help a little there; the database will need to implement an access control system of some sort.

Repudiation

- The customers using either SQL or web clients could claim not to have done things. These threats may already be mitigated by the presence of logs and log analysis. So why bother with these threats? They remind you that you need to configure logging to be on, and that you need to log the "right things," which probably include successes and failures of authentication attempts, access attempts, and in particular, the server needs to track attempts by clients to access or change logs.

Information Disclosure

- The most obvious information disclosure issues occur when confidential information in the database is exposed to the wrong client. This information may be either data (the contents of the salaries table) or metadata (the existence of the termination plans table). The information disclosure may be accidental (failure to set an ACL) or malicious (eavesdropping on the network). Information disclosure may also occur by the front end(s)—for example, an error message like "Can't connect to database foo with password bar!"
- The database files (partitions, SAN attached storage) need to be protected by the operating system and by ACLs for data within the files.
- Logs often store confidential information, and therefore need to be protected.

Denial of Service

- The front ends could be overwhelmed by random or crafted requests, especially if there are anonymous (or free) web accounts that can craft requests designed to be slow to execute.
- The network connections could be overwhelmed with data.
- The database or logs could be filled up.

- If the network between the main processes, or the processes and databases, is shared, it may become congested.

Elevation of Privilege

- Clients, either web or SQL, could attempt to run queries they're not authorized to run.
- If the client is enforcing security, then anyone who tampers with their client or its network stream will be able to run queries of their choice.
- If the database is capable of running arbitrary commands, then that capability is available to the clients.
- The log analysis program (or something pretending to be the log analysis program) may be able to run arbitrary commands or queries.

NOTE The log analysis program may be thought of as trusted, but it's drawn outside the trust boundaries. So either the thinking or the diagram (in Figure 2-1) is incorrect.

- If the DB cluster is connected to a corporate directory service and no action is taken to restrict who can log in to the database servers (or file servers), then anyone in the corporate directory, including perhaps employees, contractors, build labs, and partners can make changes on those systems.

NOTE The preceding lists in this extended example are intended to be illustrative; other threats may exist.

It is also possible to consider these threats according to the person or team that must address them, divided between Acme and its customers. As shown in Table 3-8, this illustrates the natural overlap of threat and mitigation, foreshadowing the Part III, "Managing and Addressing Threats" on how to mitigate threats. It also starts to enumerate things that are not requirements for Acme/SQL. These non-requirements should be documented and provided to customers, as covered in Chapter 12. In this table, you're seeing more and more actionable threats. As a developer or a systems administrator, you can start to see how to handle these sorts of issues. It's tempting to start to address threats in the table itself, and a natural extension to the table would be a set of ways for each actor to address the threats that apply.

Table 3-8: Addressing Threats According to Who Handles Them

THREAT	INSTANCES THAT ACME MUST HANDLE	INSTANCES THAT IT DEPARTMENTS MUST HANDLE
Spoofing	Web/SQL/other client brute forcing logins DBA (human) DB users	Web client SQL client DBA (human) DB users
Tampering	Data Management Logs	Front end(s) Database DB admin
Repudiation	Logs (Log analysis must be protected.) Certain actions from web and SQL clients will need careful logging. Certain actions from DBAs will need careful logging.	Logs (Log analysis must be protected.) If DBAs are not fully trusted, a system in another privilege domain to log all commands might be required.
Information disclosure	Data, management, and logs must be protected. Front ends must implement access control. Only the front ends should be able to access the data.	ACLs and security groups must be managed. Backups must be protected.
Denial of service	Front ends must be designed to minimize DoS risks.	The system must be deployed with sufficient resources.

Continues

Table 3-8 (***continued***)

THREAT	INSTANCES THAT ACME MUST HANDLE	INSTANCES THAT IT DEPARTMENTS MUST HANDLE
Elevation of privilege	Trusting client The DB should support prepared statements to make injection harder. No "run this command" tools should be in the default install. No default way to run commands on the server, and calls like `exec()` and `system()` must be permissioned and configurable if they exist.	Inappropriately trusting clients that are written locally Configure the DB appropriately.

STRIDE Variants

STRIDE can be a very useful mnemonic when looking for threats, but it's not perfect. In this section, you'll learn about variants of STRIDE that may help address some of its weaknesses.

STRIDE-per-Element

STRIDE-per-element makes STRIDE more prescriptive by observing that certain threats are more prevalent with certain elements of a diagram. For example, a data store is unlikely to spoof another data store (although running code can be confused as to which data store it's accessing.) By focusing on a set of threats against each element, this approach makes it easier to find threats. For example, Microsoft uses Table 3-9 as a core part of its Security Development Lifecycle threat modeling training.

Table 3-9: STRIDE-per-Element

	S	T	R	I	D	E
External Entity	x		x			
Process	x	x	x	x	x	x
Data Flow		x		x	x	
Data Store		x	?	x	x	

Applying this chart, you can focus threat analysis on how an attacker might tamper with, read data from, or prevent access to a data flow. For example, if data is flowing over a network such as Ethernet, it's trivial for someone attached to that same Ethernet to read all the content, modify it, or send a flood of packets to cause a TCP timeout. You might argue that you have some form of network segmentation, and that may mitigate the threats sufficiently for you. The question mark under repudiation indicates that logging data stores are involved in addressing repudiation, and sometimes logs will come under special attack to allow repudiation attacks.

The threat is to the element listed in Table 3-9. Each element is the victim, not the perpetrator. Therefore, if you're tampering with a data store, the threat is to the data store and the data within. If you're spoofing in a way that affects a process, then the process is the victim. So, spoofing by tampering with the network is really a spoof of the endpoint, regardless of the technical details. In other words, the other endpoint (or endpoints) are confused about what's at the other end of the connection. The chart focuses on spoofing of a process, not spoofing of the data flow. Of course, if you happen to find spoofing when looking at the data flow, obviously you should record the threat so you can address it, not worry about what sort of threat it is. STRIDE-per-element has the advantage of being prescriptive, helping you identify what to look for where without being a checklist of the form "web component: XSS, XSRF..." In skilled hands, it can be used to find new types of weaknesses in components. In less skilled hands, it can still find many common issues.

STRIDE-per-element does have two weaknesses. First, similar issues tend to crop up repeatedly in a given threat model; second, the chart may not represent your issues. In fact, Table 3-9 is somewhat specific to Microsoft. The easiest place to see this is "information disclosure by external entity," which is a good description of some privacy issues. (It is by no means a complete description of privacy.) However, the table doesn't indicate that this could be a problem. That's because Microsoft has a separate set of processes for analyzing privacy problems. Those privacy processes are outside the security threat modeling space. Therefore, if you're going to adopt this approach, it's worth analyzing whether the table covers the set of issues you care about, and if it doesn't, create a version that suits your scenario. Another place you might see the specificity is that many people want to discuss spoofing of data flows. Should that be part of STRIDE-per-element? The spoofing action is a spoofing of the endpoint, but that description may help some people to look for those threats. Also note that the more "x" marks you add, the closer you come to "consider STRIDE for each element of the diagram." The editors ask if that's a good or bad thing, and it's a fine question. If you want to be comprehensive, this is helpful; if you want to focus on the most likely issues, however, it will likely be a distraction.

So what are the exit criteria for STRIDE-per-element? When you have a threat per checkbox in the STRIDE-per-element table, you are doing reasonably well.

If you circle around and consider threats against your mitigations (or ways to bypass them) you'll be doing pretty well.

STRIDE-per-Interaction

STRIDE-per-element is a simplified approach to identifying threats, designed to be easily understood by the beginner. However, in reality, threats don't show up in a vacuum. They show up in the interactions of the system. STRIDE-per-interaction is an approach to threat enumeration that considers tuples of (*origin*, *destination*, *interaction*) and enumerates threats against them. Initially, another goal of this approach was to reduce the number of things that a modeler would have to consider, but that didn't work out as planned. STRIDE-per-interaction leads to the same number of threats as STRIDE-per-element, but the threats may be easier to understand with this approach. This approach was developed by Larry Osterman and Douglas MacIver, both of Microsoft. The STRIDE-per-interaction approach is shown in Tables 3-10 and 3-11. Both reference two processes, Contoso.exe and Fabrikam.dll. Table 3-10 shows which threats apply to each interaction, and Table 3-11 shows an example of STRIDE per interaction applied to Figure 3-1. The relationships and trust boundaries used for the named elements in both tables are shown in Figure 3-1.

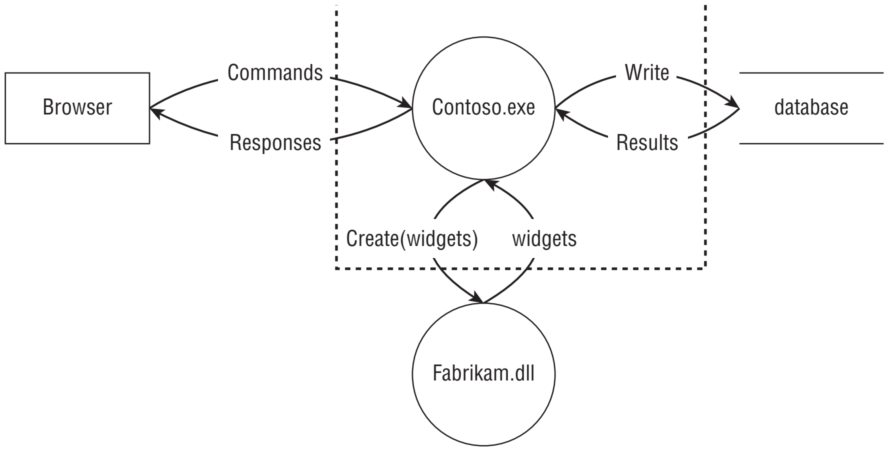

Figure 3-1: The system referenced in Table 3-10

In Table 3-10, the table columns are as follows:

- A number for referencing a line (For example, "Looking at line 2, let's look for spoofing and information disclosure threats.")

- The main element you're looking at
- The interactions that element has
- The STRIDE threats applicable to the interaction

Table 3-10: STRIDE-per-Interaction: Threat Applicability

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process (Contoso)	Process has outbound data flow to data store.	x			x		
2		Process sends output to another process.	x		x	x	x	x
3		Process sends output to external interactor (code).	x		x	x	x	
4		Process sends output to external interactor (human).			x			
5		Process has inbound data flow from data store.	x	x			x	x
6		Process has inbound data flow from a process.	x		x		x	x
7		Process has inbound data flow from external interactor.	x				x	x
8	Data Flow (com-mands/ responses)	Crosses machine boundary		x		x	x	
9	Data Store (database)	Process has outbound data flow to data store.		x	x	x	x	
10		Process has inbound data flow from data store.			x	x	x	
11	External Interactor (browser)	External interactor passes input to process.	x		x	x		
12		External interactor gets input from process.	x					

Table 3-11: STRIDE-per-Interaction (Example)

	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process (Contoso)	Process has outbound data flow to data store.	"Database" is spoofed, and Contoso writes to the wrong place.			P2: Contoso writes information in "database which should not be in database" (e.g., passwords).		
2		Process sends output to other process.	Fabrikam is spoofed, and Contoso writes to the wrong place.		Fabrikam claims not to have been called by Contoso.	P2: Fabrikam is not authorized to receive data.	None unless calls are synchronous	Fabrikam can impersonate Contoso and use its privileges.
3		Process sends output to external interactor (with the interactor being code).	Contoso is confused about the identity of the browser.		Browser disclaims and doesn't acknowledge the output.	P2: Browser gets data it's not authorized to get.	None unless calls are synchronous	
4		Process sends output to external interactor (for a human interactor).			Human disclaims seeing the output.			

Table 3-11 (continued)

ELEMENT	INTERACTION	S	T	R	I	D	E
5	Process has inbound data flow from data store.	"Database" is spoofed, and Contoso reads the wrong data.	Contoso state is corrupted by data read from the data store.			Process state is corrupted by the data retrieved from the data store.	Process internal state is corrupted based on data read from the file, leading to code execution .
6	Process has inbound data flow from a process.	Contoso believes it's getting data from Fabrikam.		Contoso denies getting data from Fabrikam.		Contoso crashes/stops due to Fabrikam interaction.	Fabrikam passes data or args that allow it to change flow of execution of Contoso.
7	Process has inbound data flow from external interactor.	Contoso believes it's getting data from the browser, when in fact it's a random attacker.				Contoso crashes/stops due to browser interaction.	Browser passes data or args that allow it to change flow of execution of Contoso.

Continues

Table 3-11 (*continued*)

	ELEMENT	INTERACTION	S	T	R	I	D	E
8	Data Flow (commands/ responses)	Crosses machine boundary		Data flow is modified by MITM attack.		The contents of the data flow are sniffed on the wire.	The data flow is interrupted by an external entity (e.g., messing with TCP sequence numbers.)	
9	Data Store (database)	Process has outbound data flow to data store.		Database is corrupted.	Contoso claims not to have written to database.	Database reveals information.	Database cannot be written to.	
10		Process has inbound data flow from data store.			Contoso claims not to have read from database.	Database discloses information.	Database cannot be read from.	
11	External Interactor (browser)	External interactor passes input to process.	Contoso is confused about the identity of the browser.		Contoso claims not to have received the data.	~~P2: process not authorized to receive the data~~ (We can't stop it.)		
12		External interactor gets input from process.	Browser is confused about the identity of Contoso.		~~Contoso claims not to have sent the data~~ (Not our problem.)			

When you have a threat per checkbox in the STRIDE-per-interaction table, you are doing reasonably well. If you circle through and consider threats against your mitigations (or ways to bypass them) you'll be doing pretty well.

STRIDE-per-interaction is too complex to use without a reference chart handy. (In contrast, STRIDE is an easy mnemonic, and STRIDE-per-element is simple enough that the chart can be memorized or printed on a wallet card.)

DESIST

DESIST is a variant of STRIDE created by Gunnar Peterson. DESIST stands for Dispute, Elevation of privilege, Spoofing, Information disclosure, Service denial, and Tampering. (Dispute replaces repudiation with a less fancy word, and Service denial replaces Denial of Service to make the acronym work.) Starting from scratch, it might make sense to use DESIST over STRIDE, but after more than a decade of STRIDE, it would be expensive to displace at Microsoft. (CEO of Scorpion Software, Dana Epp, has pointed out that acronyms with repeated letters can be challenging, a point in STRIDE's favor.) Therefore, STRIDE-per-element, rather than DESIST-per-element, exists as the norm. Either way, it's always useful to have mnemonics for helping people look for threats.

Exit Criteria

There are three ways to judge whether you're done finding threats with STRIDE. The easiest way is to see if you have a threat of each type in STRIDE. Slightly harder is ensuring you have one threat per element of the diagram. However, both of these criterion will be reached before you've found all threats. For more comprehensiveness, use STRIDE-per-element, and ensure you have one threat per check.

Not having met these criteria will tell you that you're not done, but having met them is not a guarantee of completeness.

Summary

STRIDE is a useful mnemonic for finding threats against all sorts of technological systems. STRIDE is more useful with a repertoire of more detailed threats to draw on. The tables of threats can provide that for those who are new to security, or act as reference material for security experts (a function also served by Appendix B, "Threat Trees"). There are variants of STRIDE that attempt to add focus and attention. STRIDE-per-element is very useful for this purpose, and can be customized to your needs. STRIDE-per-interaction provides more focus, but requires a crib sheet (or perhaps software) to use. If threat modeling experts were to start over, perhaps DESIST would help us make better ... progress in finding threats.