Dr. Dobbs Jolt Award Finalist 2014

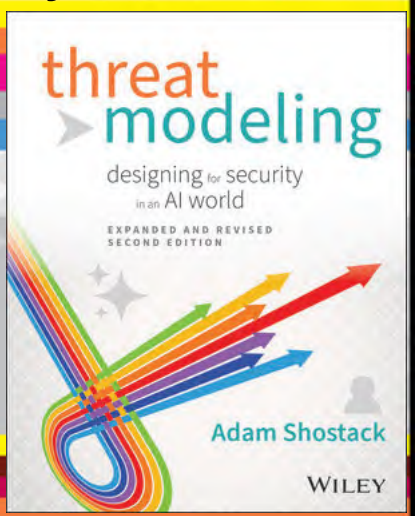
Adam Shostack
Microsoft's Threat Modeling Expert

Chapter 12 "Requirements Cookbook" from

threat modeling

designing for security

New Edition Coming January 2027



WILEY

Threat Modeling: Designing for Security

Published by
John Wiley & Sons, Inc.
10475 Crosspoint
BoulevardIndianapolis, IN 46256
www.wiley.com

Published by John Wiley & Sons, Inc., Indianapolis, Indiana
Published simultaneously in Canada

ISBN: 978-1-118-80999-0
ISBN: 978-1-118-82269-2 (ebk)
ISBN: 978-1-118-81005-7 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2013954095

CHAPTER

12

Requirements Cookbook

Important threats violate important security requirements. Ideally, those requirements are explicit, crisp, agreed-on within the development organization, and understood by customers and the people impacted by the system. Unfortunately, this is rarely the case. In part, that's because requirements are very difficult to do well. That makes requirements a tedious way to start a project, and as the agile folks will tell you, YAGNI ("you ain't gonna need it")—so we should skip straight to user stories, right? Maybe, but maybe not.

As you discover threats, you'll be forced to decide whether the threat matters. Some of that decision will be based on a risk calculation, and some will be based on a requirements calculation. If your system is not designed to maintain security in the face of hostile administrators, then all effort spent on mitigating hostile administrators will be wasted.

That said, this chapter starts with an explanation of the cookbook approach and a discussion of the interplay of requirements, threats, and mitigations. You'll then learn about ways to think about business requirements, and look at how to use common security frames to help with your requirements. (A *frame* here is a way of structuring how you look at a problem, while a *framework* is a breakdown which includes specific process steps.) The frames are "prevent/detect/respond" and "people, process, technology," with requirements in development contrasted with requirements in acquisition. Next you'll learn how to use compliance frameworks and privacy to drive your requirements. You can use these sections to decide what sort of requirements you might need.

Each of these sections may be more useful to product management than to developers. The chapter then delves deep into the more technology-centered STRIDE requirements. The STRIDE requirements are the most deeply technical and "actionable" of the requirements. You should not succumb to the temptation to make those the only requirements you consider. The chapter closes with a discussion of non-requirements.

Why a "Cookbook"?

A great many systems are available for requirements elicitation. Most of them (at best) skim over security. This chapter does not intend to replace your requirements approach, but to supplement it with a set of straw-man requirements, designed to be easily adapted to the specific needs of your system. The intent is, much like a cookbook, to give you ideas that you can easily turn into real "food." Also like a cookbook, you can't simply take what's here (the raw ingredients) and serve it to your dinner guests; but you can use what's here to prepare a scrumptious set of requirements. As you consider what you want to build, you can refer to this section for requirements that crystalize what you're thinking about.

Of course, you'll need more detail than the sample requirements can provide. For example, "Anonymous people can create/read/update/delete item" is a good starting point, but what sort of items can they create, read, update, or delete? Wikipedia has a nuanced set of answers for create, update, and delete, and another set of nuanced answers for what can be read. There's a very different set of answers on Google.com or Whitehouse.gov. The requirements examples shown cannot include the local or specific knowledge needed to make them concrete.

Most of these starting points are grouped with several related starting points. To take some examples from the section on STRIDE authentication requirements, after the requirement "Anonymous people can create/read/update/delete items," the next requirement is "All authenticated users can create/read/update/delete item," followed by "An enumerated subset of users can create/read/update/delete items." In this case, you might well need all three requirements expanded into your project's authentication requirements. In contrast, the section on authentication strength includes "We will control the authentication database" and "We will allow an outside organization (such as Facebook) to control our authentication database." These are both reasonable choices that organizations make, but you can't make both of them. The next requirement splits the difference with "We will allow an outside organization (such as Facebook) to control part of our authentication database, such as 'signed in users,' but not administrators."

The Interplay of Requirements, Threats, and Mitigations

The same sort of task interplay as discussed in the previous section takes place on a broader scale between threats, requirements, and mitigations. As threats are discovered, some of them will violate explicit requirements. Others will violate implicit requirements, offering an opportunity to improve the requirements list. Other threats may lead to discussion of whether they violate a requirement or not, again leading to possible clarification. Thus, finding threats helps you identify requirements. Discussion of threats may also lead to a discussion about the difficulty of addressing a given threat. Such discussion can also feed back into requirements when a threat can't be mitigated. Mitigations drive requirements far more often than requirements drive mitigations. (In fact, I can't think of a case where a requirement drives a mitigation without a threat, except perhaps it's a fine definition of compliance, and that may be why so many security professionals resent compliance work.) This interplay is shown visually in Figure 12–1.

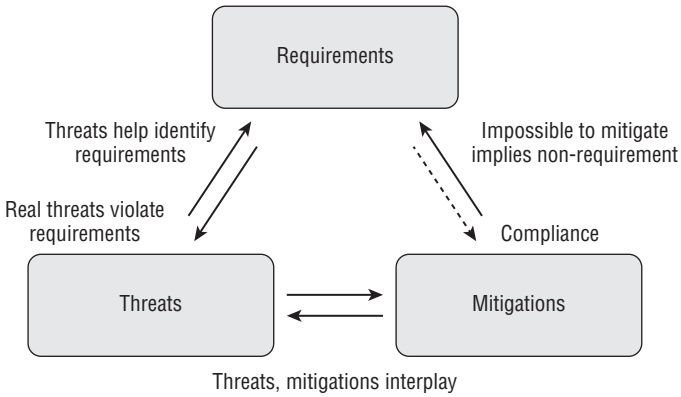


Figure 12-1: The interplay of threats, requirements, and mitigations

You might have noticed a *reductio ad absurdum* attack on this perspective. That is, taken to an extreme, it quickly becomes ridiculous. For example, if a product were to declare that the network is trusted, and all network threats are irrelevant, that would be a pretty silly decision to make, and hard to justify in front of customers if the product handles any sensitive data. However, the extreme position isn't what you should take. If you are new to threat modeling, be extremely cautious about removing requirements because you're unsure how to mitigate the threats. To check your assumption that mitigation is impossible, you should plan to spend days to weeks investigating what others have done in similar circumstances. As you develop more experience in security, that investigation will go faster, as your toolbox will be more varied.

Business Requirements

In this section you learn about the business requirements for security. These are the requirements that will make the most sense to business people. Some organizations may call these "goals," or "mission," or "vision."

Outshining the Competition

Your organization may be in a situation in which security properties or features are either a customer requirement or a competitive differentiator. In those scenarios, you can start with a requirement or requirements selected from the following list:

1. The product is no less secure than the typical competitor.
2. The product is no less secure as measured by *X* than the typical competitor.
3. The product is no less secure as measured by *X* than market leader *Y*.
4. The product will ship fewer security updates than the competition. (This can incentivize hiding vulnerabilities, which you shouldn't do.)
5. The product will have fewer exploitable vulnerabilities than the typical competitor.
6. The product will have fewer exploitable vulnerabilities than the market leader *Y*.
7. The product will be viewed as more secure than the typical competitor.
8. The product line will be viewed as more secure than the typical competitor.
9. We will be able to use security as a competitive advantage.
10. We will be able to use this security feature/property as a competitive advantage.

Industry Requirements

If your product is sold for a particular industry or use, there may be industry-specific requirements which apply. For example, if you're building payment processing software, you'll need to comply with industry rules. If you sell medical devices, you'll want to ensure that your devices have substantial defense in depth, as your ability to get them recertified quickly may be controlled by law. If you're building tools for emergency responders, they may come under particular attack. You should consider how your particular circumstances will drive your requirements.

Scenario-Driven Requirements

The use of stories, scenarios, or use cases can be a strong general approach to requirements elicitation, so it is natural to hope that security requirements might also be derived from them. That is a reasonable hope, but I am not aware of any structured approach for doing so for security requirements.

The primary challenge is that security is rarely a goal of a feature. When explaining what Alice will do as she goes about her day, few product managers say "and then we'll force her to log in again, since her session timed out over lunch" or "and then Mallory will try to break into the product by. . ." Even for security features, such as access control, it's rare to have a product manager define a scenario like "Alice adds a group to allow write access to this file, and then tests to see whether the explicit deny rule for Bob still holds."

Prevent/Detect/Respond as a Frame for Requirements

Prevent/detect/respond is a common way for organizations to think about operational security. I'm aware of a large bank that, having adopted this way of seeing the world, focuses most of its energy on response, assuming that its systems will be compromised, and focusing on reducing mean time to repair. Prevent/detect/respond can be used as a frame for thinking about requirements in development, to ensure that your technology addresses each area. It can also be used in technology acquisition or operations.

Prevention

STRIDE threat modeling is very focused on ensuring that you prevent threats. One element not well covered by STRIDE is vulnerabilities, and their management. Another is operational security approaches to prevention.

Operational Security Preventive Requirements

Operational isolation requirements focus on reducing attack surface by isolating systems from each other.

1. All production systems will be isolated from the public Internet by firewalls.
2. All production systems will be isolated from internal development and operational systems (such as HR).
3. All traversals of isolation boundaries will be authenticated by something beyond IP and port numbers.

4. All production systems will be deployed on VMs that are distinct from other production systems (that is, one application, one VM).
5. All cloud systems will be isolated from competitors. (This is easy to write, and hard to achieve.)

Least privilege requirements are focused on making it more difficult for an attacker to take advantage of an intrusion. Least privilege is easy to say and practically challenging to achieve. In the list that follows, "with privileges" means emphatically "as root/administrator," but also those points in between a "normal" user and the most privileged.

1. No production application will run with privileges.
2. No production application will run with privileges without undergoing a threat model analysis and penetration test.
3. Any production application we create that requires privileges will have those privileges isolated into a small component.
4. Application acquisition will include a discussion of threat models and operational security guidance.

Account management for prevention includes account lifecycle (as discussed in Chapter 14, "Accounts and Identity"), when it's made operational. Managing accounts across a small organization was challenging even before the days of cloud services, which make it even harder.

1. Account creation will be managed and tracked, and all service accounts will have a responsible person.
2. Accounts for people will be terminated when a person leaves.
3. Accounts will be periodically audited or reviewed to ensure that there is someone responsible for each.

Vulnerability Management

Vulnerability is a term of art that refers those accidental flaws which can be exploited by an attacker. This is in contrast to features that can be abused. In other words, a vulnerability is something that everyone agrees ought to be fixed. The exploitation of vulnerabilities can often be automated, so in order to prevent exploitation it's important to consider the vulnerability lifecycle from discovery through coding a fix, testing that fix, and delivering it. If you develop software, some of the vulnerabilities discovered will be in your code, while others will be in code on which you depend. You'll need to be able to handle both sorts. In each case, you'll need to take the fix, test it, and deliver it onwards to people in operations. Sometimes those operations people will be in the same organization,

but oftentimes they'll be external. If you're in operations, you'll need to be able to discover vulnerability reports from your suppliers, and manage them. You may also get vulnerability reports about the systems you operate, and you'll need to manage those as well.

Once you have a way to learn about the updates, you'll need a way to flow those updates through to your software and/or deployed systems. If you develop complex software, good unit tests of the functionality you use in outside components will help you rapidly test security updates and identify issues that might come with them. Maybe that's part of a continuous deployment strategy for your site, or perhaps it's something like running a security response process the way Microsoft does (Rains, 2013) so your customers can learn about, receive, and deploy your updates in their environment.

Vulnerability Reports about Your Products or Systems

Properly managing inbound vulnerability reports is an important task, and you should consider how you'll encourage people to report vulnerabilities to you. There is an alternative approach, which is "we will sue security researchers to inhibit discovery and reporting of security flaws." This backfires. For example, after Cisco sued researcher Mike Lynn for exposing flaws in its products, Cisco CSO John Stewart said "we did some silly things" and "we created a firestorm" (McMillan, 2008).

Product vulnerability management requirements might include:

1. We will have a public policy encouraging the reporting of security flaws.
2. We will have a public policy encouraging the reporting of security flaws and make it easy to do so.
3. We will have a public policy encouraging the reporting of security flaws by paying for them.
4. We will have a public policy encouraging the reporting of security flaws in our online services.
5. We will have a public policy encouraging the reporting of security flaws in our online services, and explain what testing is acceptable or unacceptable.
6. We will have automatic update functionality built into our product.
7. We will have automatic update functionality built into our product, and it will be on by default.
8. We will support only the latest version of this product, and security fixes will require an update.
9. We will support only the latest version of this product, and security fixes will require an update and those updates may include feature changes.

10. We will have a public policy on support lifetime, and produce security fixes for all currently supported products.
11. We will have a public channel for vulnerability announcements that is designed to satisfy those looking to sign up to lots of channels to track all the software they operate. As such, it will be nothing but vulnerability announcements.

Managing Vulnerabilities in External Code

It is important to track the components on which you depend and their security updates. This applies to both development and operations, and to both commercial and open-source components. Your acquisition process should include understanding how an organization notifies customers of security updates. If there's no mail list, RSS feed, or other mechanism established for security update notifications, that's probably a problem. Some possible requirements include:

1. We will only discover security issues when they're important enough for the media to talk about.
2. Each group will maintain its own list of dependencies.
3. We will maintain a single list of dependencies to track for software updates.
4. We will ensure that we track dependencies, and have a person assigned to reading the updates and generating action as appropriate.
5. Our dependency-tracking SLA will be no more than four hours from announcement to bug filed, 24 × 365.
    a. The response will be a risk assessment and possibly an action plan.
    b. The response will be to test and roll all patches of severity *X* without bothering with risk assessment.
    c. The response will be to deploy all patches and believe in our rollback practices.
6. We will maintain a testbed to roll out new patches before putting them into production.
7. We will use virtual machines taken from production to test new patches before rolling them into production.
8. We will have patch management software that can deploy to all operational services.

Operationally managing vulnerabilities is more broad than managing patches. Sometimes a vendor will release an advisory about a problem before there's a patch, and you'll have to decide if and how you want to manage such reports. An advisory may involve work in addition to or in place of patching.

Detection

Detecting security problems is a challenge in the chaos of modern operations. Even in regulated industries, most intrusions are detected by a third party. The right way to perform security logging and analysis seems to be elusive. Nevertheless, if you want to use prevent/detect/respond as a frame for thinking about requirements, useful requirements can be found in this section. There are two major goals to think about: incident detection and incident analysis. From a requirements perspective, incident detection involves logging and ensuring that the logs are analyzed. There are four main types of monitoring: change detection, signature attack detection, anomaly attack detection, and impact detection. These are discussed further in Chapter 9 "Tradeoffs When Addressing Threats" under "Wait and See." Incident analysis involves recording enough state transitions that an analyst can reconstruct what happened.

Operational requirements:

1. We will detect attacks of type *X* within time *Y*.
2. We will detect 75 percent of attacks of type *X* within time *Y*, and 50 percent of the remainder within *Z*.

Product requirements:

1. Our product will use the word "security" in all log messages that we expect are security related.
2. Our product will log in a way to help detect attacks.
3. Our product will log login attempts in a way to help detect attacks.
4. Our product will track repeated attempts to perform any action, and flag such in the logs.

There is a fuzzy line between detection and response requirements. Rigidly categorizing them probably isn't useful.

Response

Incident response teams often use an approach that mirrors the one suggested by Ripley in the movie *Aliens*, "I say we take off and nuke the entire site from orbit. It's the only way to be sure." Planning for that in product threat modeling involves a good separation between your product and its configuration and data. That separation enables the response team to nuke the product install (which may be compromised) while preserving the configuration and data (which may also be compromised, but can perhaps be cleaned up). In the same vein, publishing a set of signatures or hashes for the code you ship will help a response team check the integrity of your product after a compromise.

In operational threat modeling for response, there is a much longer set of requirements from which you can build:

1. We will have an incident response plan.
2. We will have an incident response plan in a binder on a shelf somewhere.
3. We will have an incident response plan and run annual/quarterly/monthly drills to ensure we know how to operate.
4. Our intrusion-detection SLA will be no more than four hours from detection to incident response execution, 24 × 365.
5. Our intrusion-detection SLA will be no more than eight business hours from detection to incident response execution during normal business hours.
6. Our incident response plan will [not] be designed to preserve court-quality evidence.
7. The senior administrators will be trained in our incident response plan.
8. All administrators will be trained in our incident response plan.
9. All administrators will have a wallet card with first response steps and contact information.
10. All incidents will have a lessons learned document produced.
11. All incidents will have a lessons learned document produced, appropriate to the scale of the incident.
12. Lessons learned documents will be shared with the appropriate people.
13. Lessons learned documents will be shared with all employees.
14. Lessons learned documents will be shared with all employees and partners.
15. Lessons learned documents will be published when we are required to report a breach so others can learn from our mistakes.
16. Lessons learned documents will be published so others can learn from our mistakes.

The act of publishing lessons learned documents may seem unusual, but it is increasingly common practice, and the transparency has been beneficial to business. For example, after a major outage at Amazon, they published a root cause analysis, and Netflix announced that they had used the information to improve their own service (Netflix, 2011).

CROSS-REFERENCE **See also the earlier section "Vulnerability Management."**

People/Process/Technology as a Frame for Requirements

It's also common for people to use people/process/technology as a frame for thinking about security, so it may help you to use it as a way to find requirements.

People

There are two major categories of security requirements for people: trustworthiness and skills. Trustworthiness is a matter of how much authority and discretion the people in the system are expected to have. Organizations do various levels of background checking at hiring time or as an ongoing matter. Such checks can be expensive and intrusive, and may be constrained or required by local law. If your product is intended for use by highly trusted people, you should be explicit about that. Requirements for trustworthiness can be somewhat managed by audit and retributive measures, which impose development requirements regarding logging and operational requirements around the audit process. Skill requirements should also be clearly documented. It is sometimes helpful (and usually tricky) to have a certification of some sort that attempts to assess the skills of an individual.

Security requirements focused on people might include:

1. Employees with cash management responsibility will undergo background checks for financial crimes.
2. Employees with cash management responsibility will undergo credit checks.
3. Employees with cash management responsibility will undergo regular credit checks.
4. Employees dealing with children will be required to certify that they do not have a criminal conviction.
5. Employees dealing with children will be required to certify that they do not have a criminal conviction in the last seven years.
6. Employees dealing with children will undergo a criminal background check.
7. Prospective employees will have an opportunity to contest information that is returned, as we are aware that background checks are often inaccurate.

If you are not familiar with the issues of accuracy in background checks, there are fascinating reports available, including ones from the National Consumer Law Center (Yu, 2012) or the National Employment Law Project (Neighly, 2013).

Process

Understanding how the product's technology is to be operated can also drive security requirements. Drafting a security operations guide can be a good way to elicit product security requirements.

Technology

It is very tempting to say that this entire book is about the technology, so "this section intentionally left blank." But that's not really the case. This book shows that models of your technology are the best place to start threat modeling. But all technology interacts with other technology, so what are the limits or scope of how you approach this? The best answer when you're getting started is to focus on those technologies where you can most directly address threats. As your skills grow, looking along your supply chain (and possibly the supply chains you are part of) may be a good way to expand your horizons.

Development Requirements vs. Acquisition Requirements

When you're developing technology from scratch, the set of security requirements you might choose to address is much larger than when you're acquiring technology from someone else. This issue is exacerbated by technological ecosystems. For example, the Burroughs 5500 computers of the 1960s had a memory architecture that was resistant by design to stack-smashing sorts of attacks (Hoffman, 2008; Shostack, 2008). However, it is challenging to procure a system with such features today.

In reality, few systems are really developed "from scratch." The security requirements of every project are constrained by its inputs. It is helpful to everyone along the chain to document what security requirements you do or do not support. For example, if you are developing on Windows, defending against the administrator account is not supported (Culp, 2013). Similarly, before Windows 8, you cannot defend against apps running as the same user. Windows 8 adds some capabilities in this area (Hazen, 2012).

Compliance-Driven Requirements

An ever-expanding set of security requirements is driven by compliance programs. Those compliance regimes may be imposed on you or your customers, and they make an excellent source of security requirements. More rarely, they're even a source of excellent security requirements. (It's challenging to write broad requirements that are specific enough to engineer from.) This section covers three such requirement sets: the Cloud Security Alliance (CSA) control matrix and domains, the United States NIST Publication 200, and PCI-DSS (the Payment Card Industry Data Security Standard). To the extent that you're building technology for a single organization, you may have security requirements directly imposed. If you're building technology to sell, or give away, then your customers may have requirements like these. If you expect to encounter a set of compliance requirements from your customers, it will probably be helpful to create or use a single meta-framework that covers all the issues in the frameworks you need to comply with, rather than attempt working with each one. There are commercially available "unified compliance frameworks," and depending on the number of compliance requirements you face, it may be worth buying one for speed, coverage, or expertise. The CSA framework can be a useful starting point if you need something free. The requirements that follow are presented as places from which to derive more detailed requirements. They should not be used to replace an appropriately detailed understanding of your requirements for compliance purposes.

Cloud Security Alliance

The CSA is a non-profit organization dedicated to cloud security. They have produced two documents that are helpful for security requirements: the controls domains and the controls matrix.

The first document maps controls into a set of domains, including governance and enterprise risk management, legal issues, compliance and audit, information management and data security, traditional security, business continuity and disaster recovery, data center operations, incident response, application security, encryption and key management, identity and access management, virtualization, and security as a service. This is a fine list of areas to consider, and the CSA has a lot more documentation for you to dig into. (They also consider architecture, and portability and interoperability, but those seem somewhat less relevant for security.)

The CSA also has a Cloud Control Matrix of 98 control areas, with mappings that show their applicability to architectural areas (physical, network, compute,

storage, application, and data), to corporate governance, to cloud service delivery models (SaaS/PaaS/IaaS), and to service providers versus tenants. Each control area is also mapped to COBIT, HIPAA/HITECH, ISO/IEC 27001–2005, NIST SP800–53, FedRAMP, PCI DSS, BITS Shared Assessments SIG v6 & AUP v5, GAAP, Jericho Forum, and the NERC CIP.

The requirements documented in the Cloud Controls Matrix are designed to be used as a basis for cloud operational security. Some of the requirements are most relevant to cloud services, but it's a fine resource to start with, with the advantages of being both freely available and already mapped to a large set of other sets of controls.

NIST Publication 200

Requirements in this publication are a mixed set, ranging from planning and risk assessment to physical environment protection and technical requirements, such as authorization and system integrity. Federal agencies are required to "develop and promulgate formal, documented policies and procedures . . . and ensure their effective implementation" (NIST, 2006). US Government agencies must also meet the controls laid out in NIST Special Publication 800–53. Items marked with a star align to one or more STRIDE threats, so you might cross-check the section "The STRIDE Requirements" later in this chapter.

For each item in this list, consider if there's a need to address the issue in your product requirements:

- Access control, including authorization*
- Awareness and training
- Audit and accountability including traceability of actions back to accounts*
- Certification, accreditation, and security assessments, including assessments of whether information systems are suitably protected
- Configuration management, which imposes configuration and management requirements
- Contingency planning
- Identification and authentication*
- Incident response
- Maintenance
- Motherhood and apple pie (just kidding)
- Media protection
- Physical and environment protection
- Planning, including plans for the organizational information systems and controls

- Personnel security, including criteria for who's hired, managing termination, and penalties for compliance failures
- Risk assessment of threats (including risks to mission, functions, image, or reputation) to the organization or individuals
- Systems and service acquisition, including allocating resources and deploying system development life cycles that address security
- System and communication protection*
- System and communication integrity*

PCI-DSS

The Payment Card Industry Data Security Standard (PCI-DSS) is a set of standards for those processing payment card data (PCI, 2010). These standards are generally incorporated into contracts associated with credit card processing. Note the wide variance in specificity—from "have a security policy" to "do not use default passwords"—in the following requirements:

1. Install and maintain a firewall configuration to protect cardholder data.
2. Do not use vendor-supplied defaults for system passwords and other security parameters.
3. Protect stored cardholder data.
4. Encrypt transmission of cardholder data across open, public networks.
5. Use and regularly update anti-virus software.
6. Develop and maintain secure systems and applications.
7. Restrict access to cardholder data by business need-to-know.
8. Assign a unique ID to each person with computer access.
9. Restrict physical access to cardholder data.
10. Track and monitor all access to network resources and cardholder data.
11. Regularly test security systems and processes.
12. Maintain a policy that addresses information security.

Privacy Requirements

You'll find there are a few main motivations for privacy, including legal compliance and a desire to make only those promises that can be kept—to avoid customer anger. This section covers a selection of important privacy requirements frameworks. They are important because they underlie many laws or are influential with regulators (Fair Information Practices, Privacy by Design), can

help you avoid privacy blow-ups (Seven Laws of Identity), are pragmatic, and are designed to be accessible to non-specialist software developers (Microsoft's Privacy Standards for Developers).

Fair Information Practices

Fair Information Practices (FIP) is a concept that goes back to a 1973 report for the United States Department of Health, Education, and Welfare, which put forth five core fair information Practices. Along the way, practices were promoted to principles, and if you're paying close attention, you'll notice FIP expanded to either, or sometimes even both (Gellman, 2013). The difference is minor, but I'll hew to the term used in the source discussed. The original enumeration was as follows:

1. Notice/Awareness
2. Choice/Consent
3. Access
4. Security
5. Enforcement/Redress

These form the basis for the 1980 OECD "Guidelines on the Protection of Privacy and Transborder Flows of Personal Data." The OECD put forth eight Principles. The European Union's Data Protection Directive and Canada's Personal Information Protection and Electronic Documents Act are also based on FIPs, and each has divided them into slightly different lists. These high-level principles may be a useful checklist when evaluating security and privacy issues at design time. Which list you should use will depend on a number of factors, including the location of the organization and its customer base. Be aware that many software engineers find these lists are too generic to be of much use when designing a system.

Privacy By Design

Privacy by Design is a set of principles created by the Ontario Privacy Commissioner with the goal of helping organizations embed privacy into product design. It outlines seven main principles, quoted below:

1. Proactive
2. By Default
3. Embedded
4. Positive Sum
5. Life-cycle Protection

6. Visibility/Transparency
7. Respect for Users

Privacy by Design has been criticized as "vague" and leaving "many open questions about their application when engineering systems" (Gürses, 2011).

The Seven Laws of Identity

Kim Cameron of Microsoft has put forward a set of seven principles he calls the *Laws of Identity* for digital identity systems (Cameron, 2005). They are not overall privacy requirements, but a great deal of privacy relates to how a system treats "identity," a concept further discussed in Chapter 14 "Accounts and Identity." They may be an interesting complement to contextual integrity (discussed in Chapter 6 "Privacy Tools"). These are extracted from a document that describes and contextualizes each law:

1. **User Control and Consent:** Technical identity systems must only reveal information identifying a user with the user's consent.
2. **Minimal Disclosure for a Constrained Use:** The solution that discloses the least amount of identifying information and best limits its use is the most stable long-term solution.
3. **Justifiable Parties:** Digital identity systems must be designed so the disclosure of identifying information is limited to parties having a necessary and justifiable place in a given identity relationship.
4. **Directed Identity:** A universal identity system must support both omni-directional identifiers for use by public entities and uni-directional identifiers for use by private entities, thus facilitating discovery while preventing unnecessary release of correlation handles.
5. **Pluralism of Operators and Technologies:** A universal identity system must channel and enable the inter-working of multiple identity technologies run by multiple identity providers.
6. **Human Integration:** The universal identity metasystem must define the human user to be a component of the distributed system integrated through unambiguous human-machine communication mechanisms offering protection against identity attacks.
7. **Consistent Experience Across Contexts:** The unifying identity metasystem must guarantee its users a simple, consistent experience while enabling separation of contexts through multiple operators and technologies.

These laws can be reasonably easily inverted into threats, such as "does the system get appropriate consent before revealing information?" or "does the system disclose identifying information not required for a transaction?" However,

this runs the risk of losing the richness of the Laws of Identity, so it's left as an exercise for you.

Microsoft Privacy Standards for Development

The Microsoft Privacy Standards for Developers (MPSD) is a prescriptive document that, for a set of scenarios, gives advice about how to address them. However, the standards are not focused on the discovery of privacy issues. They focus instead on a set of scenarios (notice, choice, onward transfer, access, security, and data integrity) and requirements for those scenarios (Friedberg, 2008).

The difference between these privacy standards and the more principle-oriented approaches is a result of the customer-focus. The MPSD are explicitly based on the FIPs, but are designed to provide practical advice for developers. Making it easy for (a defined) someone to use the document increases the effectiveness of an approach, and the audience-focus of the MPSD could well be emulated by other documents to help improve privacy.

Which of these privacy requirements frameworks will best inform your technology depends on what you're building, and for whom. FIPs or Privacy by Design may spark valuable discussion of your designs or goals. If your system is focused on people, the "Seven Laws" can help. If you lack privacy expertise, the MPSD will help (but it is not intended to replace professional advice).

The STRIDE Requirements

You may recall that STRIDE is the opposite of properties that you want in a system, so properly this section ought to be called "The AINCAA Requirements," but that's just not very catchy. The relationship between STRIDE and the desired properties is shown in Table 12–1.

Table 12–1: STRIDE and AINCAA

THREAT	DESIRABLE PROPERTY
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

The following subsections are organized according to the desirable property shown in Table 12–1.

Authentication

Authentication is the processes or activity which increases your confidence that something is genuine. For example, entering a password increases the system's confidence that the person behind the keyboard really is authorized to use the system.

Is Authentication Required for Various Activities?

Different systems require different levels of authentication. As discussed in the chapter introduction, many websites offer read content to anonymous people (that is, no authentication is required). Some, like Wikipedia, offer write access as well. Some requirements you can build on include:

1. Anonymous people can create/read/update/delete items.
2. All authenticated users can create/read/update/delete items.
3. An enumerated subset of users can create/read/update/delete items.

How Strong an Authentication Is Required?

As stated above, different systems require different levels of authentication, and different forms of control for the authentication system. Sample requirements include:

1. Single-factor authentication is sufficient for activity *X*.
2. Single-factor authentication plus a risk management check is sufficient for activity *X*.
3. Two-factor authentication is sufficient for activity *X*.
4. We will control the authentication database.
5. The IT/marketing/sales department will control the authentication database.
6. We will allow an outside organization (such as Facebook) to control our authentication database.
7. We will allow an outside organization (such as Facebook) to control part of our authentication database, such as "signed-in users" but not administrators.
8. Authentication should only be possible for people in IP range *X*.
9. Authentication should only be possible for people in physical location *X* (such as in the building or in the United States).

Note that systems to physically locate people are either weak, buggy (that is, high false positive, false negative rates) or exceptionally expensive (such as a dedicated air-gap network).

Account Life Cycle

The ways in which accounts are managed will vary based on your requirements. Sample requirements include:

1. Anyone can create an account.
2. Anyone with an e-mail address can create an account.
3. Anyone with a validated e-mail address can create an account.
4. Anyone with a credit card number can create an account.
5. Anyone with a valid, authorizable credit card can create an account.
6. Anyone with a credit card who can tell us how much we charged can create an account.
7. Anyone with a bank account who can tell us how much we charged can create an account.
8. Only an authorized administrator can create normal accounts.
9. Only two authorized administrators can create a new administrator account,
    a. and all other admins are notified,
    b. and an auditing team is notified. (This is perhaps a non-repudiation requirement, but you can also think of it as part of the account life cycle.)
10. Anyone can close their account at any time, and the data is deleted as soon as possible.
11. Anyone can close their account at any time, and the data is deleted after a cooling off period.
12. Anyone can close their account at any time, and the data is kept for business purposes.
13. Anyone can close their account at any time, and the data is kept for *N* time to satisfy regulatory requirements.
14. Administrator participation is required to close an account.
15. When administrator participation is required to close an account, the account must be globally inaccessible within *N* minutes.

Integrity

Sample integrity requirements include:

1. Data will be protected from arbitrary tampering.
2. This data will only be subject to modification by an enumerated set of authorized users.

3. These files/records will only be subject to modification by enumerated authorized users.
4. These files/records will only be subject to modification by enumerated authorized users, and edit actions will be logged.
5. These files/records will only be subject to modification by enumerated authorized users, and edit content will be logged.
6. These files/records will be unwritable when the system is in operation.
7. These files/records will be append-only when the system is in operation.
8. Modifications to these files/records will be cryptographically detectable.
9. Modifications to these files/records will be cryptographically detectable with commonly available tools.
10. Data sent over this inter-process channel will be protected from tampering by the operating system.
11. Data sent over this channel will be protected from tampering by a cryptographic integrity mechanism.
12. Messages will be protected from tampering by a cryptographic integrity mechanism.

You'll notice that there are integrity requirements on channels and messages. The channel is what the message travels down. Protection in the channel doesn't protect the data once it has left the channel.

For example, consider e-mail. Imagine a well-encrypted, tamper-resistant, replay-protected channel between two mail servers. The operators of those servers have confidence that the messages are well protected in that channel; but a person on one end or the other could alter a message, and forward it. If the messages themselves have tamper resistance, then that can be detected. Such tamper resistance could be imparted, for example, by a cryptographic signature scheme.

Non-Repudiation

Recall that non-repudiation can cover both business and technical requirements. Sample requirements include:

1. The system shall maintain logs.
2. The system shall protect its logs.
3. The system shall protect its logs from administrators.
4. The logs will survive compromise of a host; for example, by writing logs to a remote system.

5. The logs will allow account compromise to be differentiated from a behavior change.
6. The logs will resist counterparty fraud; for example, by using cryptographic signatures.
7. The logs will resist insider tampering; for example, with hash chains or write-once media.

Confidentiality

Sample confidentiality requirements include:

1. Data in file/database will be available only to these authorized users.
2. Data in file/database will be available only to these authorized users, even if computers/disks/tapes are stolen.
3. Data in file/database will be available only to these authorized users, even if computers are stolen while turned on.
4. The name/existence of this datastore will only be exposed to these authorized users.
5. The content of communication between Alice and Bob will only be exposed to these authorized users.
6. The topic of communication between Alice and Bob will only be exposed to these authorized users.
7. The existence of communication between Alice and Bob will only be exposed to these authorized users.

Availability

Sample availability requirements include:

1. The system shall be available 99 percent of the time.
2. The system shall be available 100 percent of the time.
3. The system shall be available 100 percent of the time, and we will pay our customers if it's not.
4. The system shall be available for *N* percent of the time, including planned maintenance.
5. Only authenticated users will be able to cause the system to spend 10× more CPU than they have spent.
6. The system will be able to resist a simple DoS such as synflooding by a 50,000-host botnet.

7. The system will be able to resist a simple DoS such as HTTPS connection initiation by a 50,000-host botnet.
8. The system will be able to resist a customized DoS by a 50,000-host botnet.

The selection of 50,000 is only somewhat arbitrary. It is large enough to be a substantial threat, while small enough that there are likely quite a few of them out there.

Authorization

Sample authorization requirements include:

1. The system shall have a central authorization engine.
2. The system shall have a central authorization engine with a configurable policy.
3. Authorization controls shall be stored with the item being controlled, such as with an ACL.
4. Authorization controls shall be stored in a central location.
5. The system shall limit who can read data at a higher security level.
6. The system shall limit who can write data to a higher integrity level.
7. The authorization engine should work with accounts or account groups.
8. The authorization engine should work with roles (properties of accounts).

There is a tension between storing authorization controls centrally versus with the objects being protected. The former is easier to manage but harder for normal people to understand (often to the point of being unsure where to go). The system controlling who can read from a higher integrity level is analogous to military data classification schemes. Someone with a secret clearance can't read a top-secret document. (This was first formalized as the Bell-LaPadula model [Bell, 1973]). Controlling who can write to a higher integrity level is also very useful, and is described by the Biba model (Biba, 1977). The Biba model is a description of how an operating system protects itself from programs running as a normal user.

Cloud and DevOps Authorization and Audit

Much of cloud operations eventually boils down to a set of authorization questions. Who is authorized to make which changes, and who made which changes. The lists might not line up, because policy and implementation don't always perfectly line up. As organizations move, intentionally or not, toward DevOps models, these questions become trickier. The requirements also change. Rather

than a formal handoff from development to test, the code is promoted to pre-production, and then to production. Sample DevOps requirements include:

1. Any developer is authorized to push code to pre-production.
2. Any developer is authorized to push code to production.
3. Any production change requires only test pass complete.
4. These production changes only are possible with test pass completion.
5. Every production change requires human signoff.
6. Every change must be designed and tested to roll back.
7. Every production change can be tracked to a person.
8. Every production change can be tracked to a test run.

Non-Requirements

Just as enumerating requirements is important, so is being explicit about what the system won't do. Some attacks are too expensive to deal with (for example, you might accept that the KGB could subvert your employees). Others may require capabilities that the operating system, chip-maker, and so on, do not currently supply. Whatever the reason, the things that your system doesn't do should be explicit, so your management, operations, or customers are not surprised. The following sections consider three ways to express and communicate non-requirements: operational guides, warnings and prompts, and Microsoft's "10 Immutable Laws of Security."

Operational Non-Requirements

Sometimes there will be things that can't be secured in the code but must be addressed in operations. The simplest example is reading the logs to detect attacks. Other goals, such as defending against malware or a malicious admin, can be attractive distractions. They generally fall outside of what you should worry about. You should document these as either requirements or non-requirements and engineer appropriately.

Have an Operational Guide

Documenting these in an operational guide serves two purposes: transparency and requirements elicitation. Transparency is useful because it helps your customers set their expectations appropriately, and avoid unpleasant surprises. The second purpose, requirements elicitation, means that as you document what an operator needs to do, you may well decide that the requirements are unrealistic,

and decide to either add features to make something more feasible, or remove features that can't be used securely.

Defend against Malware in the Right Way

Trying to defend a system against malware that is already on the system is a complex and probably futile effort for most products. The exceptions are operating systems creators and security software creators. If you're not in either group, and if the malware is running inside the same trust boundaries as your code, there's little you can do.

Most software systems should focus on preventing the elevation of privilege threats that allow malicious software to run. You should assume that the system is working on behalf of the people authorized to run code. (Obviously, this doesn't apply if you're creating an operating system or anti-virus product.)

Decide if Defending against Admin Is a Requirement

Most systems should not try to defend against the malicious admin. (The same principle as malware applies, but worse.) If the admin of a system is malicious, then they can do a wide variety of things. Some cryptographic systems may allow your data to transit these systems safely, but if you decrypt data on a system controlled by a malicious admin, they can capture your password or the plaintext of your documents. (Worse, they can get tricky. For example, telling you that a good cryptographic key didn't work, tricking you into entering other passwords you commonly use.)

If you're creating a high-assurance system of some form, you might have a requirement to always apply a two-person rule to defend against administrators. If that's the case, then you have a fine challenge ahead.

Warnings and Prompts

Some things that can't be secured in the code are sufficiently dangerous that there should be a warning before allowing the system to proceed. These things can be considered non-requirements, or they can be used to drive requirements that improve the architecture of the system. See Chapter 15 "Human Factors and Usability" for more information.

Microsoft's "10 Immutable Laws"

Microsoft's "10 Immutable Laws of Security" are an example of how to explain what your system doesn't do. The second paragraph of that document opens with "Don't hold your breath waiting for an update that will protect you from

the issues we'll discuss below. It isn't possible for Microsoft or any software vendor to 'fix' them, because they result from the way computers work" (Culp, 2013). The first several "laws" are as follows:

> Law #1: If a bad guy can persuade you to run his program on your computer, it's not solely your computer anymore.
>
> Law #2: If a bad guy can alter the operating system on your computer, it's not your computer anymore.
>
> Law #3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.
>
> Law #4: If you allow a bad guy to run active content in your website, it's not your website anymore.
>
> Law #5: Weak passwords trump strong security.
>
> Law #6: A computer is only as secure as the administrator is trustworthy.

You might note that laws 1, 2, and 4 sure do seem like slight variations on one another. That's to draw out the implications.

Summary

In this chapter, you've learned that good security requirements act as a complement to threat modeling, enabling you to make better decisions about threats you discover with the techniques in Part II of this book.

You've been given a set of base requirements that are designed to help you do the following:

- Understand the space of security requirements better.
- Quickly crystalize more precise requirements.

This chapter should serve as a practical, go-to resource when you're working through requirements at a business or technology level. The business level includes requirements driven by competitive pressure and industry, and requirements to handle vulnerabilities that your code might contain.

You've also seen how to use people/process/technology and prevent/detect/respond to inform your requirements process. Compliance frameworks, including those from the Cloud Security Alliance, the U.S. government, and the payment card industry can be used as a base for your requirements.

You've learned about a variety of sources for privacy requirements. Lastly, you've been reminded how the STRIDE threats violate properties, and how those properties can be developed into requirements.