Dr. Dobbs Jolt Award Finalist 2014

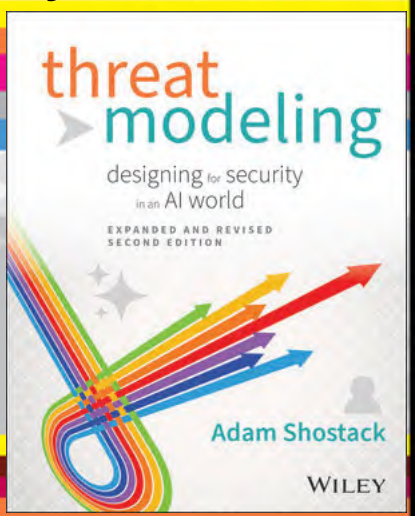
Adam Shostack
Microsoft's Threat Modeling Expert

Chapter 18 "Experimental Approaches" 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

18

Experimental Approaches

Today's approaches to threat modeling are good enough that a wide variety of people with diverse backgrounds and knowledge can use them to find threats against systems they are developing, designing, or deploying. However, there's no reason to believe that current approaches are the pinnacle of threat modeling. The same smart people who are finding new ways to reconceptualize programming and operations will find new ways to approach threat modeling.

This chapter presents some promising approaches with one or more identifiable issues to overcome. Those issues can include a lack of success with the method when used by those other than its inventors or a lack of prescriptiveness. Those approaches include looking in the seams; operational threat modeling approaches, including the FlipIT game and kill chains; the Broad Street taxonomy; and adversarial machine learning. This chapter also discusses threats to threat modeling approaches, risks to be aware of as you create your own techniques or approaches, and closes with a section on how to experiment.

Some of these approaches are like Lego building blocks, and can easily be attached to modeling software with DFDs and STRIDE, while others take a different approach to a problem, and are harder to snap together. The approaches that can be plugged into other systems include a discussion about how you can do that.

Looking in the Seams

You can find threats by bringing teams together to discuss the design of their software, and using the resultant arguments as a basis for investigation. The premise is that if two teams have different perspectives on how their software works, then it's likely the software has seams that an attacker could take advantage of. This technique has been around for quite a long time, and is routinely borne out in conversations with experts (McGraw, 2011). Why then is it listed under experimental? Because there is limited advice available about what to do to actually make it work.

One team at Microsoft has produced a methodology it calls *intersystem review* (Marshall, 2013). This methodology was designed to build on DFDs and STRIDE, and it works well with them. The questions (listed below) can probably be applied to other approaches. What follows is a version of the intersystem review process, edited to make it more generally applicable. Thanks to Andrew Marshall of Microsoft for agreeing to share the work which forms the basis for this section.

The participants in an intersystem review are development, test, and program management contacts for both (or all) sides of the system, along with security experts. Before meeting in person, someone responsible for security should ensure that the threat models for each team are documented, and pay close attention to the external dependency lists. Unless otherwise noted/decided, that responsible person should ensure that each step in the process, described as follows, is completed.

The terms *product* and *product group* are intended to be interchangeable with "service," and the approach here may even be applied between companies, agencies, or other entities. The first two steps are assigned to the product group, as they are most likely to understand their own systems.

1. For each system, the product group should document data obtained from other products:
    a. For what purpose or purposes is data validated?
    b. What purposes/use cases/scenarios are known not to be supported?
    c. Is there an assumption that specific validation or tests will be performed by the receiver? (If so, is that in the developer documentation and any sample code?)
    d. Does inbound data have any particular storage, security, or validation concerns associated with it?
    e. What edge cases are developers and testers concerned about?

2. For each system, the product group should document data sent to other products:
   a. What promises does the product make about content, format, integrity or trustworthiness of the data?
   b. What purposes is the data suitable for?
   c. What validation is the recipient expected to perform? Where are those requirements documented?
   d. What expectations are there for privacy or data protection by the receiver?
   e. What edge cases are developers and testers concerned about?
3. Create a system model including each system and the trust boundaries. The model can be imperfect, and imperfection may provoke discussion.
4. Have the cross-system group meet in person. The agenda should include:
   a. If the participants are new to intersystem-review, explain the goals of the process and meeting.
   b. A walk-through of the diagrams and scenarios each component supports
   c. Document design, coding, and testing assumptions made by each team on data received by dependencies. Outside the meeting, these assumptions can be checked.
   d. Deep-dive into edge cases, especially around error handling and recovery.
   e. Review prior security bugs specific to services or interfaces exposed at the trust boundaries.

Depending on the relationship between the systems involved, it may be helpful to pre-define a decision model with respect to bugs identified during the meeting. Such a decision model is especially important if participants may suggest or demand bugs be filed or addressed in code other than their own, and if that otherwise wouldn't be the case. For example, participants might not have that ability if the seams are between two organizations. Also helpful is an assigned note taker or recording of the meeting.

Operational Threat Models

The models described in this section, FlipIT and kill chains, are designed to be of value to people operating systems. They span a gamut, from the deeply theoretical approach of FlipIT to the deeply practical kill chains.

FlipIT

FlipIT is a game created by Ari Juels, Ron Rivest, and colleagues. (Ari is Chief Scientist for RSA, Inc., and Ron is one of the creators of the RSA cryptosystem.) FlipIT is played by two players, each of whom would like to control an IT system for as long as possible. Each can, at any time, and at some cost, check to see if they control the system, and if they do not, take control. Whoever controls the system at a given time is earning points, but the score is hidden until the end of the game (otherwise, it would leak information about who's in control). The object of the game is to have more points than your opponent at the end. Perhaps obviously, FlipIT is more a game in the sense of the Prisoner's Dilemma than a game like Monopoly. To help you get a sense for the game, there is a simple online demonstration (Bowers, 2012), and it has a certain charm and ability to pull you into playing.

FlipIT is a model of an IT system and an intruder. Each would like to control the system at a minimal cost. Its authors have used FlipIT as a model to demonstrate how password changes can be made more secure at a lower cost. I'm optimistic that FlipIT can be effectively used to model a system's security in additional interesting ways. FlipIT is listed as experimental because to date only its authors have used it to find new insights.

FlipIT is a very different sort of model compared to other forms of operational threat modeling, and if it's possible to integrate it with other approaches, how to do so is not yet clear.

Kill Chains

The concept of "kill chains" comes from analysis at the US Air Force. There have been several attempts to apply *kill chain* approaches to operational threat modeling. The essential idea of a kill chain is that most attacks involve more than simply taking over a computer or gathering usernames and passwords via phishing. There is a chain of events that includes such steps, but as technology exploitation becomes commercialized and weaponized, understanding the chain of activity gives defenders more opportunities to interfere with it. These kill chain models seem to benefit from customization to align with the mental models of defenders who are using them.

The models in a kill chain approach model both the actions of the attackers and the reactions of the defenders. The idea was introduced in a paper "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains," by Eric Hutchins and colleagues at Lockheed Martin (Hutchins, 2011). You'll see these referred to as

"LM kill chains" in this section. There is also work from Microsoft on "threat genomics" which is closely related and discussed below.

Kill chains are fairly different from other threat elicitation approaches. They are like Erector Sets while STRIDE is like Legos. There might be an opportunity to use the defensive technologies as a bridge to other defensive techniques or tools, but it could be awkward.

LM Kill Chains

The LM kill chain paper presents the idea of using kill chains to drive defensive activity. The authors' approach models attacker activity in terms of indicators. These indicators are either atomic, computed, or behavioral. An atomic indicator is one that cannot be further broken down while retaining meaning. A computed indicator is also derived from data in the incident (rather than an interpretation), and can take forms such as a regular expression. A behavioral indicator is created by combining atomic and computed indicators, and might be a sentence designed for a person to read and consider.

The LM kill chain model outlines seven phases that attackers go through:

- **Reconnaissance:** Research, identification and selection of targets
- **Weaponization:** Combining an exploit and remote access tool into a package for delivery
- **Delivery:** Delivering the package to the target. LM reports that e-mail, websites, and USB media were most commonly observed.
- **Exploitation:** Some action to trigger intruder code, often via a vulnerability in an OS or application (See also the section "The Broad Street Taxonomy," later in this chapter.)
- **Installation:** Installing the remote access tool into the targeted system
- **Command and Control (C2):** Establishing and using a communications channel with the attacker
- **Actions on Objectives:** The actual work that motivates all of the above

The model also posits that there are defensive actions that a defender can take, and includes a table (redrawn as Table 18-1) with an information operations doctrine of detect, deny, disrupt, degrade, deceive, and destroy. The defensive doctrine is derived from the U.S. military. The acronyms used are: IDS (Intrusion Detection System), NIDS (Network IDS), HIDS (Host IDS), NIPS (Network Intrusion Prevention System), and DEP (Data Execution Protection).

Table 18-1: LM Courses of Action Matrix

PHASE	DETECT	DENY	DISRUPT	DEGRADE	DECEIVE	DESTROY
Recon	Web analytics	Firewall ACL				
Weaponize	NIDS	NIPS				
Deliver	Vigilant User	Proxy Filter	In-Line AV	Queuing		
Exploit	HIDS	Patch	DEP			
Install	HIDS	"chroot" Jail	AV			
C2	NIDS	Firewall ACL	NIPS	Tarpit	DNS Redirect	
Actions on Objectives	Audit Log			Quality of Service	Honeypot	

Source: Hutchins, et al. 2011.

The paper usefully shows how indicators can provide a way to look earlier and later in the chain to find other aspects of an intrusion, and how common indicators may show that multiple intrusions were made by the same attacker. Taking data from other phases to find places to look for indicators of attack is an important way to model and focus defender activity.

Threat Genomics

Another approach that shows promise is Espenschied and Gunn's threat genomics (Espenschied, 2012). This work models the detectable changes that attackers introduce into an operational system. In contrast to the LM model, the threat genomics model focuses on detectable changes, rather than operational steps, that an attacker would progress through. The approach aims to build a model of an attack from those changes, and then apply the models to improve detection and predictive capabilities. Threat genomics models are a set of what the authors call *threat sequences*. A sequence is a set of state transitions over time. The states are as follows:

- Reconnaissance
- Commencement
- Entry
- Foothold
- Lateral movement
- Acquired control

- Acquired target
- Implement/execute
- Conceal and maintain
- Withdraw

Note that the states are not sequential, and not all are required. For example, an attack that installs a remote access tool may involve entry and foothold as the same action. Steps such as reconnaissance or lateral movement may not be required at all. (This is in contrast to the LM model.) The sequences are based on observable indicators, such as log entries. Note that this model "sits above" many of the "security indicators" systems, such as OpenIOC, STIX, and so on. The sequences are intended to move analysts away from interpreting individual indicators or correlations to interpreting correlations between sequences.

After an attack, if investigators piece together enough elements of the sequence, then the sequence and/or its details may provide information about an attacker's tools, techniques, and procedures. If these are graphed, then different graphs may help to distinguish between attackers. A sample sequence from Espenschied and Gunn's paper is shown in Figure 18-1.

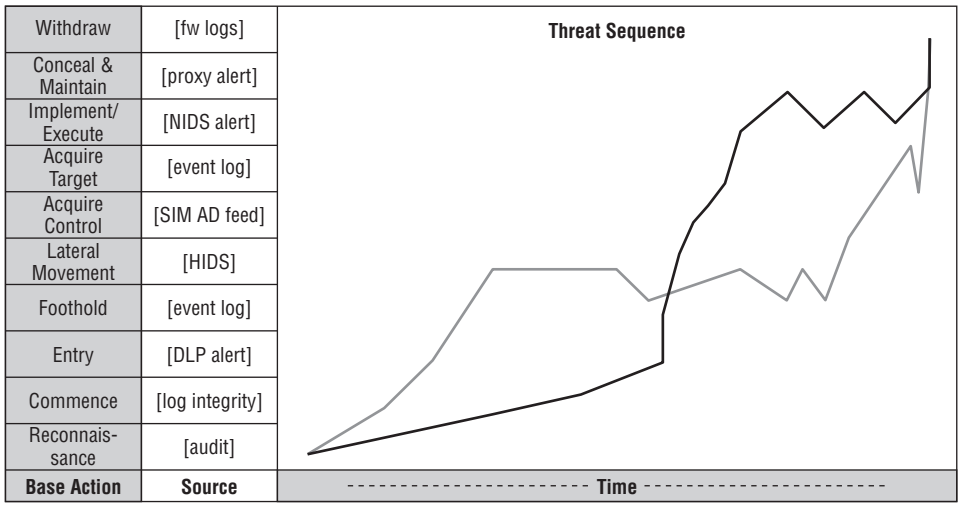


Figure 18-1: Threat genomics example

The sequences model enables an investigator to understand where indicators should be present. For example, before a target can be acquired, the attacker has to enter and establish a foothold.

This model is also a helpful way to consider what data sources would detect which state transitions. For example, domain controller change reports may help discover control acquisition, but they will not directly help you find the initial

point of entry. Figure 18-2, taken from the paper, shows a sample mapping of data sources to transitions.

Data Source	1	2	3	4	5	6	7	8	9	10
Web server logs		■	■							
Email server logs		■	■	■						
DB logs				■			■	■		
TwCSec assessment				■		■	■	■		
VLAN logs					■				■	■
AD domain change reports					■	■			■	
OS Windows event logs		■	■							
OS other desktop logs			■							
AV logs		■	■	■		■				■
Host scan logs				■		■				
HIDS		■				■				
ACS/FEP event logs			■				■			
Web proxy logs	■	■		■		■		■	■	
IDS/IPS logs	■	■	■	■	■	■		■	■	■
Firewall logs		■								

Figure 18-2: Mapping data sources to transitions

If a column is missing, you should consider whether that transition is something you want to detect. The list of possible rows is long. If a defensive technology row doesn't match any column, it's worth asking what it's for, although the failure to find a match may result from it simply not lining up well with the threat genomics approach. The defensive technologies shown should be taken as examples, not a complete list. Threat genomics is listed as experimental because to date (as far as I know) only its authors have made use of it.

The "Broad Street" Taxonomy

I developed the "Broad Street Taxonomy" and named it after a seminal event in the history of public health: Dr. John Snow's identification of a London street water pump as the source of contamination during an outbreak of cholera in 1854. Not only did he demonstrate a link, but he removed the handle of the water pump, and in doing so probably altered the course of the epidemic. For more on the history of that event, see *The Ghost Map* by Steven Johnson (Penguin, 2006). It is both a taxonomy, that is a system for categorizing things, and a model, in that it abstracts away details to help focus attention on certain aspects of those events. I chose the name Broad Street to focus attention on the desire to understand

how computers come to harm, and it enables activities to address those causes. The use of an aspirational, rather than a functional name, was also driven by the fact that the taxonomy categorizes a set of things that are somewhat tricky to describe. Part of the reason they are tricky to describe is that the taxonomy groups them together for the first time. By analogy, before Linnaeus built a tree of life, vertebrae referred only to backbones, not the set of creatures with backbones. By categorizing living organisms, he created a new way of seeing them. (My aspirations are somewhat less . . . broad.)

So what does Broad Street model? The taxonomy is designed to clarify how computers are actually compromised ("broken into") for malware installations, and it has shown promise for use in incident root cause analysis. It focuses only on issues that have been repeatedly documented in the field. The taxonomy helps understand compromises in a way that can effectively drive product design and improvement. The value of the model is its focus on the means of compromise for an important set of compromises. However, Broad Street is neither a model of compromise nor a model of how malware gets onto systems. It's not a model of compromise because it doesn't touch on compromises that start with stolen credentials. Nor is it a complete model of how malware gets onto systems, as it excludes malware that is installed by other malware.

NOTE **The model has had a deep impact at Microsoft, resulting in an update to AutoPlay being shipped via Windows Update, but its use has been limited elsewhere (so far).**

Before getting into the taxonomy itself, note that Broad Street does not align well with software development threat modeling; its model of the world is too coarse to be of use to most developers, who consider particular features. Thus, continuing with the toy analogies, Broad Street is like Lincoln Logs.

When represented as a taxonomy, Broad Street includes a set of questions that are designed to enable defenders to categorize an attack in a consistent way. The questions are ordered, and presented as a flowchart, as shown in Figure 18-3. The questions are designed to be applied to a single instance of compromise, or, in the case of malware that uses several different approaches to compromise a system, serially across each technique, resulting in each technique having a label. The questions are explained after the flowchart. The simplified presentation in a flowchart is easy to use, but many nodes have nuances that are hard to capture in the short labels. The exit condition for categorizing an attack is that an attack must have a label. Note that in Figure 18-3, there are boxes labeled "I'm unsure" and "hard to categorize." These are intended for those using the taxonomy to record those problems. (Figure 18-3 shows version 2.7 of the taxonomy.)

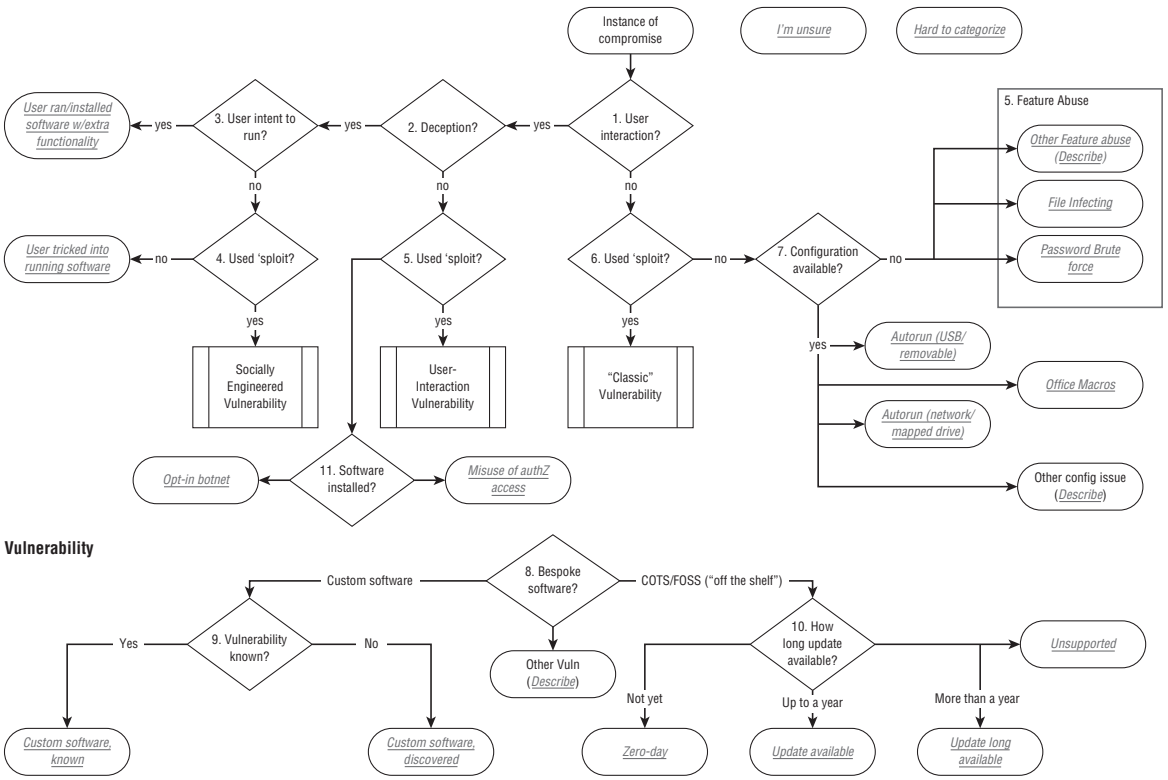


Figure 18-3: The Broad Street Taxonomy

1. **User interaction?** The first question the taxonomy poses is whether a person has to perform some action that results in a compromise. Asked another way, if no one is logged into the computer, can the attack work? If the answer is yes, the flow proceeds to question 2; if no, the flow proceeds to question 6.
2. **Deception?** The second question is one of deception. Deception often entails convincing someone that they will get some benefit from the action, or suffer some penalty if they don't do it, using any of a variety of social engineering techniques. (Table 15-1 provides ways to describe such techniques.) Examples of deception might include a website telling people that they need to install a codec to watch a video, or an e-mail message that claims to be from the tax authorities. There are a variety of actions that a "normal person" will believe are safe, such as browsing well-known websites or visiting a local file share. (Earlier variants of the taxonomy addressed this by asking, "Does the user click through a warning of some form?" and while that is still a good criterion, it is hard to argue that removing warnings from software should lead to changing the way something is categorized.) If propagation requires deceiving the victim, the flow proceeds to question 3. If it doesn't, question 3 is skipped and the flow proceeds to question 5.
3. **User intent to run (software)?** If interaction is required, is the person aware that the action they are taking will involve running or installing software? If the answer is yes, the incident can be categorized by the endpoint: *User ran/installed software (with unexpected functionality)*. The person runs the software, which does unexpected and malicious actions in addition to, or instead of, the software's desired function. A significant overlap exists between this and the traditional definitions of Trojan Horse software. The analogy with the Trojan Horse from Greek mythology refers to the way a lot of malware gains access to victims' computers by masquerading as something innocuous: malicious programs represented as installers for legitimate security programs, for example, or disguised as documents for common desktop applications. This label can cause two types of confusion. First, it could lead to multiple endpoints with the same label. Second, many security vendors define "trojan" [sic] as a program that is unable to spread of its own accord.
4. **Used 'sploit?/Deserves a CVE?** These questions have the same intent. Different presentations of the taxonomy use different presentations of this question, selected to be more usable for a particular audience. This question has the same meaning for nodes (4, 5, and 6) of the process flow, and determines whether or not a vulnerability is involved. Because the term "vulnerability" can be open to interpretation, the question asks whether the method used to install the software is of the sort often documented

in the Common Vulnerabilities and Exposures list (CVE), a standardized repository of vulnerability information maintained at `cve.mitre.org`. ("Deserves" is used to cover situations in which the method meets the CVE criteria but has not yet been assigned a CVE number, as with a previously undisclosed vulnerability. However, the CVE does not cover less frequently deployed systems, so "deserving of a CVE" may be read as "a thing which would get a CVE if it were in a popular product.") This question can also be read as "Was an exploit used?" (where exploit refers to a small piece of software designed to exploit a vulnerability in software, and often written as "sploit"). If question 4 is answered no, then the incident is categorized: *User tricked into running software*. This result indicates a false badging, such as a malicious executable named document.pdf.exe with an icon similar or identical to the one used for PDF files in Adobe Reader. The victim launches the executable, believing it to be an ordinary PDF file, and it installs malware or takes other malicious actions. If question 4 is answered yes, then you can categorize the means of compromise as *socially engineered vulnerability*, and possibly further categorize it through the vulnerability subprocess (nodes 8, 9 and 10).

5. **Deserves a CVE?** If question 5 is answered yes, then you can categorize it as a *user-interaction vulnerability*, and possibly further categorize it through the vulnerability subprocess. The taxonomy does not use the popular "drive-by-download" label because that term is used in several ways. One is analogous to these issues; the others are what are labeled: *User runs/installs software with extra functionality* and *User tricked into running software*. If it does not deserve a CVE, then you can refer to the endpoint as an *Opt-in botnet*, a phrase coined by Gunter Ollman (Ollman, 2010). In some cases, people choose to install software that is designed to perform malicious actions. For example, this category includes Low Orbit Ion Cannon (LOIC), an open-source network attack tool designed to perform DoS attacks.
6. **Deserves a CVE?** If question 6 is answered yes, then you should categorize it as a *classic vulnerability*, and possibly further categorize it through the vulnerability subprocess.
7. **Configuration Available?** Can the attack vector be eliminated through configuration changes, or does it involve intrinsic product features that cannot be disabled through configuration? Configuration options would include things like turning the firewall off and using a registry change to disable the AutoRun feature. If the answer is yes—in other words, if the

attack vector can be eliminated through configuration changes—the flow terminates in one of four endpoints:

a. **AutoRun (USB/removable)**: The attack took advantage of the AutoRun feature in Windows to propagate on USB storage devices and other removable volumes.

b. **AutoRun (network/mapped drive)**: The threat takes advantage of the AutoRun feature to propagate via network volumes mapped to drive letters.

c. **Office Macros**: The threat propagates to new computers when victims open Microsoft Office documents with malicious macros.

d. **Other configuration issue**: This catch-all is designed to accumulate issues over time until we can better categorize them.

If the answer is no—in other words, if the attack vector uses product features that cannot be turned off via a configuration option—then the vector is called *feature abuse*, which includes three subcategories:

a. **File infecting viruses**: The threat spreads by modifying files, often with .exe or .scr extensions, by rewriting or overwriting some code segments. To spread between computers, the virus writes to network drives or removable drives.

b. **Password brute force**: The threat spreads by attempting brute-force password attacks—for example, via ssh or rlogin or against available SMB volumes to obtain write or execute permissions.

c. **Other feature abuse**: This is another catch-all, designed to accumulate issues over time until we can better categorize them.

8. **Bespoke Software Project?** This question is designed to distinguish locally developed software from widely available software. Vulnerabilities are not unique to commercial (or open-source) software, and other exploit analyses have found that vulnerabilities in custom software, such as website code, account for a significant percentage of exploitation (Verizon, 2013).

9. **Vulnerability known?** This question serves to distinguish between issues discovered by the owner/operator/creator of the software and those found by an attacker. For vulnerabilities discovered by an organization, there is some period of time between discovery and patching while the vulnerability is reproduced, and code is fixed and tested. One endpoint here, *Custom software, known (to owner)*, is for that set. The other

endpoint, *Custom software*, *discovered (by attacker)*, is for those vulnerabilities first found by an attacker.

10. **How long update available?** (The abbreviated text fits in a box, and means "how long has the update been available to install?"):
    a. **Zero-day**: Refers to a vulnerability for which no patch was available from the software creator at the time of exploitation.
    b. **Update available**: Refers to a vulnerability for which a patch was available from the software creator for up to a year at the time of exploitation.
    c. **Update long available**: Refers to a vulnerability for which a patch was available from the software creator for over a year at the time of exploitation.
    d. **Unsupported**: Refers to a vulnerability in the software that the creator no longer supports, including when the creator is out of business.

The Broad Street Taxonomy is a way to categorize and model an important set of things which are not otherwise brought together. Modeling those things in a new way has helped to improve the security of systems, and that modeling and categorization is worth exploring in additional areas.

Adversarial Machine Learning

Because machine learning approaches help solve a wide variety of problems, they have been applied to security, some in authentication, others in spam or other attack detection spaces. Attackers know this is happening, and have started to attack machine learning systems. As a result, academics and defenders are starting to examine a security subfield called *adversarial machine learning*. In a paper of that name, the authors propose a categorization with three properties (Huang, 2011). The properties are influence, security goals, and attacker goals. The influence property includes attacks against the training data, and exploratory attacks against the operational system. The second property describes what security goal is violated, including integrity of the detector's ability to detect intrusions; and availability, meaning the system becomes so noisy that real positives cannot be detected. The security goals also include privacy, meaning attacks that compromise information about the people using the system, and it appears to be a subset of information disclosure attacks. The third category is a spectrum of attacker goals, from targeted to indiscriminate.

The paper lays out the taxonomy including descriptions of further attacks in detail, walks through a number of case studies, and discusses defenses. (False negatives are included in their bulleted explanation of availability, but not in the text. Their taxonomy may be more clear if you treat induction of false negatives

as an integrity attack, and caused false positives as an availability attack.) This is a field that is likely to explode over the next few years, as more people believe that big data and machine learning will solve their security problems.

Adversarial machine learning can be contextualized as a second-order threat that is relevant when machine learning is used as a mitigation technique. Currently, there is no clean model demonstrating how machine learning can help mitigate threats, which inhibits saying exactly where adversarial machine learning will help or hurt.

Threat Modeling a Business

How to threat model something bigger than a piece of software or a system being deployed is a fair question, and one that security and operations people would like to be able to address in consistent, predictable ways that offer a high return on investment.

There appears to be tension between scope and the value that organizations receive for their threat modeling investments. That is, threat modeling more specific technologies is easier than threat modeling something as large and complex as a business. Perhaps at some level, all organizations are similar? At another level, each one has unique assets and threats against those assets. The most mature system for modeling a business is OCTAVE-Allegro from CERT-CC.

OCTAVE is the Operationally Critical Threat, Asset, and Vulnerability Evaluation approach to risk assessment and planning. There are three, interlinked methods: the original; an OCTAVE-S method for smaller organizations; and OCTAVE-Allegro, which is positioned as a streamlined approach. All are designed for operational risk management, rather than development time, and all focus on operational risks.

The methodology is freely available from the CERT.org website, and it is clearly organized into a set of phases and activities, with defined roles and responsibilities. The free materials include worksheets, examples and a book. Training classes are also offered. It is one of the more fully developed methodologies available, and those looking to create a new approach should examine OCTAVE and understand why each element is present.

OCTAVE Allegro consists of eight steps organized into four phases:

1. Develop risk measurement criteria and organizational drivers.
2. Create a profile of each critical information asset.
3. Identify threats to each information asset.
4. Identify and analyze risks to information assets and begin to develop mitigation approaches.

Risks are to be brainstormed (or approached with a provided threat tree) in six areas, each of which has an associated worksheet:

- Reputation and customer confidence
- Financial
- Productivity
- Safety and health
- Fines/legal penalties
- User-defined

The documentation notes that "working through each branch of the threat trees to identify threat scenarios can be a tedious exercise." This is an ongoing challenge for all such methodologies, and one that offers a real opportunity for helping a large set of organizations if someone finds a good balance here. OCTAVE and its family do not interconnect in obvious ways with other methods. Perhaps this family of systems is like Revel model airplanes. If the kit is what you need, it's what you need, but it may be a little tedious to put it together?

Threats to Threat Modeling Approaches

Henry Spencer said, "those who don't understand Unix are condemned to reinvent it, poorly." The same applies to threat modeling. If you don't understand what has come before, then how can you know if you're doing something new? If you know you're doing something new but you're changing training, tasks, or techniques at random, how can you expect the outcome to be better? If you don't understand the issues in what came before, how do you know if you're tweaking the right things?

There are a number of common ways to fail at threat modeling. The first is not trying, which is self-evident. Some additional important ones are discussed in this section. They are broken into dangerous deliverables and dangerous approaches.

Dangerous Deliverables

These are two outputs which tend to lead to failure. The first is to create an enumeration of all assumptions (made worse by starting with that list), the second is threat model reports.

Enumerate All Assumptions

The advice to "enumerate all assumptions" is common within threat modeling systems, yet it is full of fail. It's full of fail for a number of reasons, including

that enumerating all assumptions is impossible; and even if it were possible, it would be an unbounded and unscoped activity. It's also highly stymieing. Let's start with why enumerating all assumptions is not possible.

Using a simple example, I could write, "I assume readers of this book speak English." That sentence contains a number of assumptions, such as that this book will be read, and that it will be read in English. Both assumptions are, in part, false. I could assume that reading, in the sense I'm using it, incorporates an audio-book (hey, an author can hope, or at least look forward to text-to-speech improving). It also incorporates the assumption that the book won't be translated. Underlying the word "English" is the strange belief that there exists a definable thing that we both call the English language, and that each of the terms I use will be read by the reader in the manner in which I intend it. You could reasonably argue that those assumptions are silly and irrelevant to our purposes. You could do so even if you were steeped in arguments over what a language is, because our goal here is to discuss threat modeling; and to the extent that we're within the same communities of practice and considering threat modeling as a technical discipline, you'd be right. However, threat modeling is not a single community of practice. Experts in different things are expert in different things, and the assumptions they make that matter to one another are not obvious in advance. That's why they are assumptions, rather than stated. Therefore, asking people to start a threat modeling process by enumerating assumptions is going to stymie them.

Even though enumerating all assumptions is impossible, tracking assumptions as you go can be a valuable activity. There are a few key differences that make this work. First, it's not first—that is, it doesn't act as an inhibitor to getting started. Second, it relies on documenting what you discover through the natural flow of work. Third, assumptions are often responsible for issues falling between cracks, so investigating and validating assumptions often pays off. (See the discussion of *intersystem review* in the "Looking in the Seams" section earlier in this chapter for advice on teasing out assumptions from large systems.)

Threat Model Reports

Threat modeling projects have a long and unfortunate history of producing a report as a final deliverable. That's because very early threat modeling was done by consultants, and consultants deliver reports. Their customers turn those into bugs, or perhaps more commonly, shelfware. Reports are not, in and of themselves, bad. A good threat analysis can be a useful input to requirements, can help software engineers think about problems, and can be a useful input into a test plan. Good notes to API callers or non-requirements can help things not fall through the seams. A good analysis might turn the threats into stories so they stay close to mind as software is being written or reviewed. This is an area where attacker-centric modeling may help. A good story contains conflict, and conflict has sides. In this case, you are one side, and an attacker is the other side.

Dangerous Approaches

Sometimes, looking at what you should not do can be more instructive than looking at what you should do. This section describes some approaches to threat modeling that share a common characteristic: They are all ways to fail.

- **Cargo culting:** The term *cargo cult science* comes to us from Richard Feynman:

> *In the South Seas there is a cargo cult of people. During the war they saw airplanes with lots of good materials, and they want the same thing to happen now. So they've arranged to make things like runways, to put fires along the sides of the runways, to make a wooden hut for a man to sit in, with two wooden pieces on his head to headphones and bars of bamboo sticking out like antennas—he's the controller—and they wait for the airplanes to land. They're doing everything right. The form is perfect. It looks exactly the way it looked before. But it doesn't work. No airplanes land. So I call these things cargo cult science, because they follow all the apparent precepts and forms of scientific investigation, but they're missing something essential, because the planes don't land.*
>
> from *Surely You're Joking, Mr. Feynman!*
> (Feynman, 2010)

  The complexities of threat modeling will sometimes combine with demands by leadership to result in cargo-cult threat modeling. That is, people go through the motions and try to threat model, but they lack an understanding of the steps, completing them by rote. If you don't understand why a step is present, you should eliminate it, and see if you get value from what remains.

- **The kitchen sink:** Closely related to cargo culting is the "kitchen sink" approach to the threat modeling process. These systems are sometimes developed by adherents of several approaches who need to work together. No one wants to leave out their favorite bit, and either no one wants to make a decision or no one is empowered to make it stick. The trouble with the kitchen sink approach is that effort is wasted, and momentum toward fixing problems can be lost.
- **Think like an attacker:** The advice to think like an attacker is common, and it's easy to repeat it without thinking. The problem is that telling most people to think like an attacker is like telling them to think like a professional chef. Even my friends who enjoy cooking have little idea how a chef approaches what dishes to put on a menu, or how to manage a kitchen so that 100 people are fed in an hour. Therefore, if you're going to ask people to think like an attacker, you need to give them supports,

such as lists of attacker goals or techniques. Effort to become familiar with those lists absorbs "space in the brain" that must be shared with the system being built and possible attacks (Shostack, 2008b). "Think like an attacker" may be useful as an exhortation to get people into the mood of threat modeling (Kelsey, 2008).

- **You're never done threat modeling:** Security experts love to say that you're never done threat modeling, and there are few better ways to ensure that you're never going to get it included in a project plan. If you can't schedule the work, if you can't describe a deliverable that fits into a delivery checklist, then threat modeling is unlikely to be an essential aspect of delivery.
- ***This* is the way to threat model:** Another idea that hurts threat modeling is the belief that there's one right way to do it. One outgrowth of this is what might be called the "stew" model of threat modeling: just throw in whatever appeals, and it'll probably work out. Similarly, too much advice on threat modeling currently available is not clearly situated or related to other advice, and as such there is often an implicit stew approach. If you select ingredients from random recipes on the Internet, you're unlikely to make a tasty stew; and if you select ingredients from random approaches to threat modeling, do you expect anything better? Good advice on threat modeling includes the context in which an approach, methodology, or task is intended to be used. It also talks about prerequisites and skills. It is made concrete with a list of deliverables, but more important, how those deliverables are expected to be used.
- **The way I threat model is...**: Every approach that has been criticized in this book has not only advocates, but advocates who have successfully applied the approach. They are likely outraged that their approach is being questioned, and with good reason. After all, it worked for them. However, that's no guarantee that it will work for others. One key goal of this book is to provide structured approaches to threat modeling that can be effectively integrated into a development or operations methodology in a cost-effective way. A useful approach to threat modeling will scale beyond its inventor.
- **Security has to be about protecting assets:** This is so obvious a truism that it's nearly unchallengeable. If you're not investing to protect an asset, why are you investing? What is the asset worth? If you're investing more than the value of the asset, why do it? All of these are great questions, and well worth asking. It's hard to argue with the importance of either. If you have no assets to protect, don't invest in threat modeling. At the same time, these questions aren't always easy to answer. Modeling around assets is

easier with operational systems than with "boxed" software (although the software-centric approach does tend to work well for operational systems). The importance of the question, however, isn't always aligned with when it should be asked, or even with the project at hand. "What gives meaning to your life?" is an important question, but if every software project started with that question, a lot of them wouldn't get very far.

How to Experiment

After reading about all the ways that threat modeling can go astray, it can be tempting to decide that it's just too hard. Perhaps that's true. It is very hard to create an approach that helps those who are not expert threat modelers, and it's very hard to create an approach that helps experts—but it is not impossible. If you have an understanding of what you want to make better, and an understanding of what has failed, many people will make something better, something that works for their organization in new ways. To do that, you'll want to define a problem, find aspects of that problem that you can measure, measure those things, introduce a change, measure again, and study your results.

Define a Problem

The first step is to know what you're trying to improve. What is it about the many systems in this book and elsewhere that is insufficient for your needs? Why are they not working? Define your goal. You may end up solving a different problem than you expect, and that may be OK; but if you don't know where you're going, you're unlikely to know if you've arrived.

Developing a good experiment around threat modeling is challenging. Perhaps more tractable is interviewing developers or surveying them after a task. Knowing what you're trying to improve can help you decide on the right questions to ask. Are you trying to get threat modeling going? In that case, perhaps ask participants if they think it was worthwhile and how many bugs they filed. Are you comparing two systems to find more threats? Are you trying to make the process run faster? See how long they spent, and how many bugs they filed. Knowing what you're trying to achieve is a key part of measuring what you're getting.

Find Aspects to Measure and Measure Them

It's easy to make changes and hope that they have the appropriate results. It can be harder to experiment, but the best way to understand what you've done

is to structure an experiment. Those experiments can be narrow, such as creating better training around STRIDE, or broader, such as replacing the four-stage model. At the core of an experiment should be some sort of testable hypothesis: If we do *X*, we'll get better results than we do with *Y*.

Designing a good experiment involves setting up several closely related tests, with as few variances between them as possible. Therefore, you might want to keep the system used in the test the same. You might want to use the same people, but if they're threat modeling the same system twice, then data from one test might taint the other. Putting people through multiple training sessions might show a different result from putting them through one (in fact, you'd hope it would). Therefore, you might bring different people in, but how do you ensure they have similar skills and backgrounds? How do you ensure that what you're testing is the approach, rather than the training? Perhaps one training has a better-looking presenter, or show more enthusiasm when covering one approach versus another. You should also review the advice on running user tests in Chapter 15 "Human Factors and Usability."

Study Your Results

If you've done something better, how much better is it? What's the benefit? Does it lead you to think that the line of inquiry is complete, or is there more opportunity to fix the issues that are causing you to experiment? What will it cost to roll it out across the relevant population? Can the new practices coexist with the old, or will they be confusing? These factors, along with the size and dispersion of an organization, influence the speed and frequency of new rollouts.

When you've built something new and useful, you should give it a name. Just as you wouldn't call your new programming language "programming language," you shouldn't name what you created "threat modeling." Give it a unique name.

Summary

There are many promising approaches to threat modeling, and a lot of ways in which experimentation will improve our approaches. Knowing what has been done and what has failed are helpful input to such experiments.

The first promising approach is to look in the seams between systems, and this chapter gives you a structured approach to doing so. You also looked at a few other approaches to operational threat modeling that show promise, including the FlipIT game and two kill chain models. There is also a Broad Street Taxonomy, which is designed to help understand bad outcomes in the real world. Lastly, there is an emergent academic field studying adversarial

machine learning, which will be an important part of understanding when machine learning systems can help you mitigate threats.

This chapter also examined a long list of threats to threat modeling approaches, including dangerous deliverables (lists of assumptions and threat model reports) and dangerous approaches. The dangerous approaches include cargo culting and throwing in everything but the kitchen sink. They also include exhorting people to "think like an attacker"; telling them (or yourself) "you're never done threat modeling"; saying "the way to threat model is"; or "the way I threat model is"; and the ever-popular distraction, "security has to be about assets."

Knowing all of this sets you up to innovate and experiment. You should do so for a problem that you can clearly articulate and for which you can measure the results of an experiment (as challenging as that can be).