Software Design Misuse and Abuse Cases
Getting Past the Positive

Paco Hope

Gary McGraw, Ph.D.
Features Drive Development
Software development is all about making software do something. People who build software tend to describe software requirements in terms of what a system will do when everything goes right. The focus is on functionality. When software vendors sell products, they talk about what those products do to make customers’ lives easier: improving business processes or doing something else positive.

Following this trend, most systems for describing software tend to describe features. UML, use cases, and other modeling and design tools allow software professionals to formalize what the software will do. They try to describe in a concrete way all the positive actions a system will take.

Consider a payroll system that allows the human resources department to control salaries and benefits. A use case might say “The system shall allow users in the HR management group to view and modify salaries of all employees.” It might even go so far as to say “The system will only allow a basic user to view his or her own salary.” These are direct statements of what the system will do.

Savvy software practitioners are beginning to think beyond features, touching on emergent properties of software systems such as reliability, security, and performance. This is due in large part to the fact that more-experienced customers are beginning to say “We want the software to be secure” or “We want the software to be reliable.” Note that, in many of these instances, it is still up to the software developer to define “secure” and “reliable” and then create secure and reliable software.

The problem is that security, reliability, and other software “-illities” are complicated. In order to create secure and reliable software, we must anticipate abnormal behavior. We don’t typically describe this behavior in use cases. Nor do we describe it with UML. However, we must have some way of talking about and preparing for abnormal behavior.

To make this concept concrete, think about a potential attacker. An attacker is likely to actively try to gain extra privileges in the payroll system and remove evidence of a fraudulent transaction. Similarly, an attacker might try to delay all the paychecks by a day and embezzle the interest that is accrued during the delay.

“Misuse” (or “abuse”) cases are a tool that can help an organization begin to view their software in the same light that attackers see it. By getting past the positive features and contemplating negative events, software security professionals can better understand how to create secure and reliable software.

Security is Not a Set of Features
Security is not a feature that can be added to software. There is no convenient “security” pulldown-menu feature where security is selected and magic happens. Unfortunately, many software producers mistakenly rely solely on cryptography somewhere in their software, assuming that security needs are addressed everywhere. Product literature often makes feature-based claims about security such as “Uses SSL,” or “includes 128-bit encryption.” While true, these statements too often represent the entirety of the vendor’s approach to securing their product. This is natural and forgivable, but it is still a problem.

Security is an emergent property of a system, not a feature. This is similar to how “being dry” is an emergent property of being inside a tent in the rain. The tent only keeps people dry if the poles are stabilized, vertical and able to support the weight of wet fabric, etc. Likewise, the tent must have waterproof fabric that has no holes and is large enough to cover all the people who want to remain dry. Lastly, all the people who want to be dry must remain under the tent the entire time it is raining. While it is important to have poles and fabric, it is not enough to say “the tent has poles and fabric: it keeps you dry!” That, however, is analogous to the claims software vendors make when they highlight numbers of bits and encryption algorithms. It is true that cryptography of one kind or another is usually necessary in order to create a secure system, but cryptography alone is not sufficient for building secure software.
Because security is not a feature, it can’t be “bolted on” after other software features are codified. Instead, security must be built in from the ground up, showing up as a critical part of the design from the very beginning (requirements development) and being included in every subsequent phase of development all the way through the fielding of the complete system. If the requirements call for the software to be “easy to use,” which translates into requiring access without passwords, then adding passwords and other authentication mechanisms after the program is completed can be very problematic. If the encryption libraries are present but not used in all the right places, adding correctly applied cryptography throughout a product can be time-consuming and costly.

The best, most cost-effective approach, then, incorporates thinking beyond the positive feature set and doing so throughout the entire development process. Every time a new requirement, feature or use case is created, someone should spend some time thinking about how that feature might be abused. Such thinking should be augmented by the advice of professionals who know how features are attacked and how best to protect software.

Thinking About What You Can't Do

Attackers are not standard-issue customers. They are bad people with malicious intent who want your software to act in an unanticipated way: to their benefit. An attacker’s goal is to think of something you didn’t think of and exploit it in a way you didn’t expect—to their gain and probably to your detriment. If the development process doesn’t address unexpected or abnormal behavior, then an attacker usually has more raw material with which to work.

Attackers are creative. Despite their creativity, however, we can be sure that certain well-known locations will always be probed in the course of their attacks: boundary conditions, edges, inter-system communication, system assumptions. Clever attackers always try to undermine the assumptions on which a system was built. For example, if a design assumes that connections from the web server to the database server are always valid, an attacker will try to make the web server send inappropriate requests to the database server in order to access valuable data. If a design assumes that web browser cookies are never modified by the client before they are sent back, attackers will intentionally cause problems by modifying cookies. Building Secure Software¹ teaches us to be mindful of the unintended ways in which our assumptions can be undermined.

The original designers of a system are in a great position to know systems better than potential attackers do. We must leverage this knowledge to the benefit of security and reliability and can do so by asking and answering the following critical questions: What assumptions are implicit in our system? What kinds of actions or conditions can render our assumptions false?

Unfortunately, those who create a system rarely make the best security analysts for that system. It is extremely difficult to consciously note and consider all assumptions (especially thinking like an attacker). Fortunately, these professionals make excellent subject matter experts (SMEs) who can be powerfully combined with security professionals. Together SMEs and security analysts can ferret out base assumptions in a system under analysis and think through the ways an attacker will approach the software.

Creating Useful Misuse Cases

The simplest, most practical method for creating misuse cases is usually through a process of informed brainstorming. While there are a number of theoretical methods that involve fully specifying a system with rigorous formal models and logics, such activities are extremely time- and resource-intensive. The good news is that they are often unnecessary in the real world. A more practical approach involves forming brainstorming teams that combine security and reliability experts with SMEs. This approach tends to cover a lot of ground quickly and yield more useful results in less time. This approach relies heavily on experience and expertise.

To guide the brainstorming, software security experts ask many questions that help identify the places where the system is likely to have weaknesses. This activity mirrors the kind of thinking that an attacking adversary performs.

Misuse is always possible at the places where legitimate use is possible. Thus, the brainstorming involves a careful look at all user interfaces (including environmental factors) and considers actions most developers assume a person can't or won't do? Assumptions about these can’ts and won’ts take many forms, such as: “Users can’t enter more than 50 characters because the JavaScript code won't let them” or “The user won’t understand the format of the cached data. They can’t modify it or use it.” These are exactly the kinds of can’ts and won’ts that attackers can and will make happen.2

Of course, all systems are vulnerable to attack in more places that just their obvious front doors. Where can a bad guy be positioned? On the wire? At a workstation? In the back office? Any communications line between two endpoints is a place where an attacker can try to interpose himself. What can a bad guy do? Watch network traffic? Modify and replay network traffic? Read files stored on the workstation? Change registry keys or configuration files? Be the DLL? Be the “chip”?

One of the goals of misuse cases is to decide and document a priori how the software should react to illegitimate use. The process of specifying abuse cases makes a designer differentiate appropriate use from inappropriate use very clearly. Approaching this problem involves asking the right questions. For example, how can the system distinguish good input from bad? How can the system tell that a request is coming from a legitimate Java applet and not from a rogue application replaying traffic? Trying to answer questions like these is likely to cause a software designer to question design and architecture assumptions. So much the better. This puts the designer squarely ahead of the attacker by fixing a problem before it ever was created!

But No One Would Do That!
System architects and project management often respond to the very idea of misuse cases by claiming “but no one would do that.” Interestingly, these claims are correct if the worldview is limited to legitimate users. Virtually any system that has value, however, can be abused. Few systems truly operate securely in a free-for-all permissions system, despite how much trust management may seem to place on the users. This problem is exacerbated by the rush to move software into a network-based (web services) model. Limiting system activity to legitimate users may be possible on a secure proprietary network, but it is categorically impossible on the Internet. The fact is that malicious users do exist in both kinds of environment, but thwarting a significant portion of them is often straightforward.

Abuse Cases in Practice
Cigital recently reviewed a client-server application that manipulated a financially sensitive database. The architecture was set up so that the server trusted its client-side application to manage all of the permissions. No permissions were enforced on the server itself. In fact, only the client had any notion of permissions. The client program ran on garden-variety desktop PCs and downloaded entire tables of data—most of which the particular user was not allowed to see—to the user’s hard disk. Herein lies an attack.

If the user looked in the application’s cache and used an unzip utility, they could see all sorts of information they should not be allowed to see. Who would do that, though? Someone who wants to manipulate the system intentionally.

It turns out that the client also enforced which messages were sent to the server, and the server honored messages independent of the user’s actual credentials. The server was assuming that any messages coming from the client had passed the client software’s permissions system and were, therefore, legitimate. By intercepting network traffic, corrupting values in the client software’s cache, or building a hostile client, malicious users could inject data into the database that they should not be able to read (much less write).

2 It is not uncommon for Cigital consultants to hear the refrain “But you can’t do that!” when shown a vulnerability or exploit. Getting to the can’ts and won’ts involves getting past the positive.
Clearly, generating misuse cases is important. The main benefit of misuse cases is that they provide essential insight into a system’s assumptions and how those assumptions will be approached by attackers. Of course, like all good things, misuse cases can be overused (and generated forever). A solid approach to this technique requires a combination of security experts and SMEs to prioritize the misuse cases as they are generated and to strike the right balance between cost and value.

Gary Mcgraw, Ph.D., is the Chief Technology Officer at Cigital, Inc. He can be reached at gem@cigital.com.

Paco Hope is a Senior Software Security Consultant at Cigital, Inc. He can be reached at paco@cigital.com.

Additional References

About Cigital, Inc.
Founded in 1992 on the simple, compelling premise that software must work, Cigital has become a major provider of software quality management (SQM) solutions. Grounded in research and proven in practice, Cigital’s SQM Solutions help companies drive down the costs of producing quality software. Cigital’s experts, technology and in-depth industry experience enable companies to repeatedly develop and deliver better software faster. Cigital solutions use a combination of process frameworks, technology tools, knowledge assets and project assessments to assure, audit and improve the quality of software products and associated development processes. The Company is headquartered in Northern Virginia. For more information on Cigital, please visit http://www.cigital.com.