Editor: **Jane Cleland-Huang**
DePaul University
jhuang@cs.depaul.edu

Keeping Ahead of Our Adversaries

Jane Cleland-Huang, Tamara Denning, Tadayoshi Kohno, Forrest Shull, and Samuel Weber

EVERY SOFTWARE SYSTEM is potentially vulnerable in ways that aren't always imagined during development. For example, pacemakers and implantable cardioverter-defibrillators (ICDs), which monitor and regulate cardiac rhythms, typically provide wireless access to healthcare providers so that they can modify settings and collect telemetry. However, a malicious user could transmit commands to ICDs to collect private data or change the device's therapy settings.[1,2] Recently, well-known hacker Barnaby Jack claimed to have developed software that let him shock patients within a 50-foot radius. Anticipating such potential threats, doctors proactively disabled the wireless features of former US Vice President Dick Cheney's pacemaker.

Malicious attacks' potential to cause real (and diverse) harm holds true for numerous other software systems. For example, University of Michigan researchers demonstrated how easy it was to take control of a traffic light system: a person could ensure that the lights were always green along his or her route or could seriously disrupt traffic by turning all the lights red.[3] Similarly, University of California, San Diego and University of Washington researchers used a car's telematics unit to remotely disable the brakes, turn off the headlights, and manipulate dashboard gauges.[4,5]

White-collar crime involving data breaches are rampant, and governments are investigating the potential for terrorist attacks on power grids, airplanes, and other public services. Technology is a double-edged blade: although computers let us pursue ever-more-impressive innovations, we're likewise subjected to growing possibilities for abuse.

So, how do we build secure products that are hardened against adversarial attacks? Let's take a look.

Thinking about Threats

Many steps to improve security can be taken at various stages of software development. However, an important place to begin is with a dedicated analysis of potential threats. Without a sound understanding of the possible threats against a given system, it's unlikely that developers will be able to adequately defend against them.[6] Surprisingly, this step is often performed hastily or skipped. One problem is that developers often assume they understand all common attack patterns and therefore fail to explore each system's specific vulnerabilities. Alternatively, they might assume they can patch in security later in the design by following accepted security policies and procedures.

To some extent, each system is unique. Even supposing that the system components are well understood and previously have been composed in the same way, the ways a deployed system is used, misused, or reappropriated can introduce unanticipated security vulnerabilities. Building a secure system requires proactive, rigorous analysis of the threats to which it might be exposed, followed by systematic transformation of those threats into security-related requirements. These requirements can then be tracked throughout the development life cycle.

Threat modeling aims to

- identify attackers' potential abilities and goals and
- catalog possible threats that the system must be designed to mitigate.

We consider threat modeling a requirements activity. The most benefit comes from understanding what security requirements are needed and

 0740-7459/16/$33.00 © 2016 IEEE

using those requirements to drive architecture decisions, develop test strategies, and engage in other software development activities.

However, in reality, threat-modeling techniques vary and can be applied to both existing and greenfield systems; different techniques are more suited to different software development activities and different development domains. Some techniques are like checklists, enumerating possible threats developers should consider in the context of their system. Others are less deterministic and try to inject more creativity to stimulate thinking about unusual attack vectors. All techniques encourage developers to think more critically about their system and about ways to subvert it; this contrasts with the more usual approaches that focus on functionality. As you might expect, developers find it exceptionally difficult to be complete and consistent and to truly put themselves in the shoes of an attacker.

Security Cards: A Threat Brainstorming Toolkit

To assist threat analysis, Tamara Denning, Batya Friedman, and Tadayoshi Kohno developed the Security Cards.[7] The Security Cards consist of 42 cards divided into four categories, or dimensions: Human Impact, Adversary's Motivations, Adversary's Resources, and Adversary's Methods.

Here, we illustrate how the cards might serve as starting points to explore potential threats to a technological system—in this case, an ICD. This thought exercise is only to explore what these software development processes would look like when applied to a system concept. We don't intend to make statements regarding current ICD security or what security considerations have been incorporated into the development process.

Human Impact

This dimension explores how security breaches could affect humans. The impacts range from personal-privacy violations to widespread societal impact. Threat-modeling sessions could start by ranking the Human Impact cards according to their relevance to the system under consideration. In this case, a highly relevant card is the Physical Well-being card (the first card in Figure 1). It asks us to think about how a misused or compromised ICD could impact people's physical well-being. However, we could also consider cards such as Emotional Wellbeing (for example, patients are aware of the threat to their health), Financial Wellbeing or Relationships (for example, the attack aims to discredit

FIGURE 1. Four Security Cards. Developers can use Security Cards to explore potential threats to a technological system—in this case, an implantable cardioverter-defibrillator (ICD).

the ICD company), or Personal Data (for example, the attacker wants to use the identifying personal data stored on the device).

Adversary's Motivations

This dimension explores why someone might want to attack a system. It helps provide a framework to explore a potential attack's scope and intended targets. For example, the Malice or Revenge card (the second card in Figure 1) might lead us to consider the situation in which an adversary attacks the ICD user owing to extreme emotion. Other motivation-related cards could include Self-Promotion (for example, the attacker wants to demonstrate technical prowess) or Diplomacy or Warfare (for example, the attacker aims to take down a political enemy who happens to have an ICD). Considering potential adversaries' motivations helps determine what resources they might have and helps us construct attacker profiles.

Adversary's Resources

This dimension explores assets an adversary might use to launch an attack. These include hardware and software tools, technical expertise, and various forms of influence. In this case, we select the Expertise card (the third card in Figure 1) and consider the hacker's potential technical skills. Another relevant card could be A Future World, which considers potential future attacks, given that interest exists in increasing the capabilities of remote checkups. We might also consider Impunity (for example, the attack might be difficult to pin on a particular person or to prosecute) or Inside Knowledge (for example, a former employee uses detailed, proprietary knowledge about the architecture).

Adversary's Methods

This dimension explores how an adversary might attack the system, including technology, coercion, and abusing logistical and bureaucratic processes. We might select the Technological Attack card (the fourth card in Figure 1), given that researchers have previously demonstrated such attacks. We also could consider cards such as Multi-Phase Attack (for example, the adversary tampers with software in the doctor's office responsible for sending commands to the ICD), Indirect Attack, or Attack Cover-up.

From Threats to Requirements

Exhaustively cataloging threats is of limited use if we don't use the information to improve the software we're developing. To illustrate moving from threats to requirements, suppose our threat model contains the following threat, written from a malicious user's perspective: "As an IT specialist with intent to physically harm an ICD patient, I'll launch an attack on the device that will change the intended effects on the patient's heart."

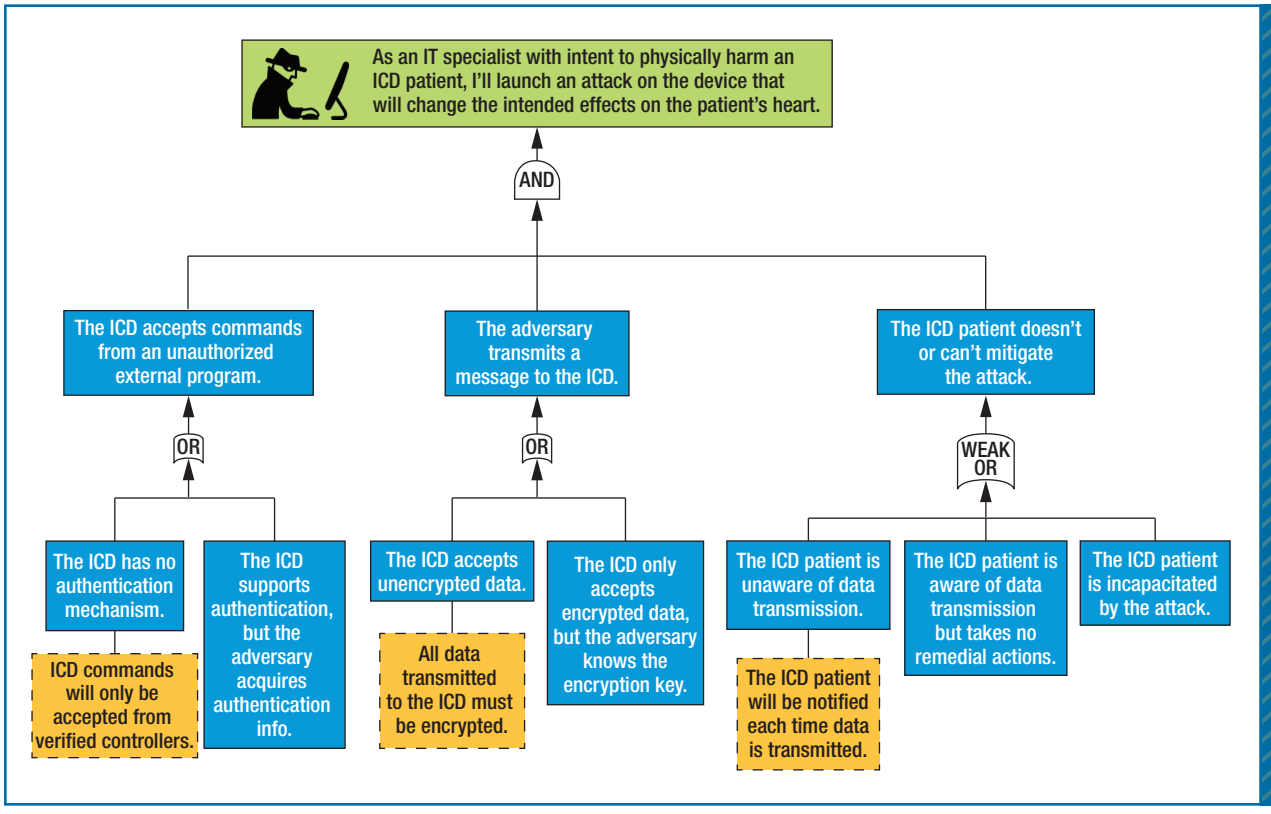

FIGURE 2. A threat tree models system vulnerabilities that potentially enable the threat—in this case, an attack on an ICD. Additional vulnerabilities could exist that the figure doesn't show.

We need to identify and specify requirements that prevent this adversary from achieving this goal. The first step is to identify vulnerabilities that enable each specific threat.

Figure 2 illustrates this through a partial analysis of vulnerabilities and issues that might facilitate an attack on an ICD. The ICD is vulnerable if it lacks an authentication mechanism or the adversary acquires authentication by stealing login information or eavesdropping. To execute the attack, the adversary must successfully transmit a valid command to the ICD. For the attack to succeed, the ICD patient should either be unaware of the attack and therefore unable to take remedial action (such as moving out of transmission range) or be immediately incapacitated.

We then analyze the associated vulnerabilities, evaluate possible mitigations, and specify them as candidate requirements. The scenario in which the ICD has no authentication mechanism is a potential problem for embedded medical devices, in which power consumption is crucial. To address this problem, we consider specifying the requirement, "ICD commands will be accepted only from verified controllers." To address the scenario in which the attacker acquires authentication, we consider specifying the requirement, "All data transmitted to the ICD must be encrypted." However, we must carefully examine both requirements and balance them against the need for additional processing, which would drain battery life. Furthermore, decreased accessibility could inhibit access to the ICD in an emergency.

So, we might consider an alternate requirement. In lieu of limiting access to verified controllers and encrypting data, a next-best option might be to provide an audible warning to ICD patients each time the device starts communicating with a controller. Such a requirement would clearly be a tradeoff. It's unlikely to provide sufficient security in the face of Barnaby Jack's shock attack, for example, but it might partially protect the user from privacy invasions or unauthorized reconfigurations.

Maintaining Traceability

Security requirements are driven by the threat-modeling process but are ultimately constrained by hardware and software tradeoffs. However, being able to maintain traceability from specific requirements back to the threats they address will likely be useful as tradeoffs among requirements are negotiated. It's important to ensure that some mitigation for important threats is maintained, even if the form of that mitigation needs to adapt and evolve.

Luckily, most of us don't have to build systems that must resist attacks by determined nations and whose failure would cause people to die. Attempting to provide perfect security for such systems not only isn't necessary but also almost certainly wouldn't be cost effective. Another benefit of threat modeling (and according to some people, the primary benefit) is documenting what threats won't be mitigated.

Without a documented, consistent understanding of what threats are out of scope, systems typically end up with extremely poor security. For example, many stories exist of systems whose password reset functionalities totally undermined all the other security features. After all, an attacker only needs to target the weakest point in a system's defenses. As another example, many organizations decide that it's not worth building technical solutions to counter insider attacks (employees deliberately doing malicious actions to their employer's systems). However, without explicitly documenting this decision, it's all too easy to overlook that the system must be built so that after employees have been fired, their knowledge of system passwords and procedures can't be used against their ex-employer.

In the end, we all agree that security requirements are needed. However, writing them without engaging in threat modeling will likely lead to cookie-cutter requirements that capture the same old problems. We will probably remember to include standard security functions, although we might not remember to specify them in the requirements document. It's less likely we'll think about specific threats that might be unique to our system. So, threat modeling is an essential activity that should form a natural prelude to the requirements process.

Acknowledgments

This material is based partly on work funded and supported by the US Department of Defense under contract FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References

1. D. Halperin et al., "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses," *Proc. 2008 IEEE Symp. Security and Privacy* (SP 08), 2008, pp. 129–142.
2. S. Gollakota et al., "They Can Hear Your Heartbeats: Non-invasive Security for Implanted Medical Devices," *Proc. ACM SIGCOMM 2011 Conf.* (SIGCOMM 11), 2011, pp. 2–13.
3. B. Ghena et al., "Green Lights Forever: Analyzing the Security of Traffic Infrastructure," *Proc. 8th USENIX Workshop Offensive Technologies* (WOOT 14), 2014; www.usenix.org/conference/woot14/workshop-program/presentation/ghena.
4. A. Czeskis et al., "Experimental Security Analysis of a Modern Automobile," *Proc. 2010 IEEE Symp. Security and Privacy* (SP 10), 2010, pp. 447–462.
5. S. Checkoway et al., "Comprehensive Experimental Analyses of Automotive Attack Surfaces," *Proc. 20th USENIX Conf. Security* (SEC 11), 2011, p. 6.
6. A. Shostack, *Threat Modeling: Designing for Security*, John Wiley & Sons, 2014.
7. T. Denning, B. Friedman, and T. Kohno, *The Security Cards: A Security Threat Brainstorming Toolkit*, Univ. Washington, 2013; http://securitycards.cs.washington.edu.

JANE CLELAND-HUANG is a professor of software engineering at DePaul University. Contact her at jhuang@cs.depaul.edu.

TAMARA DENNING is an assistant professor at the University of Utah's School of Computing. Contact her at tdenning@cs.utah.edu.

TADAYOSHI KOHNO is the Short-Dooley Professor in the Department of Computer Science & Engineering and an adjunct associate professor in the Information School at the University of Washington. Contact him at yoshi@cs.washington.edu.

FORREST SHULL is the assistant director for empirical research at Carnegie Mellon University's Software Engineering Institute. He's editor in chief emeritus of *IEEE Software*. Contact him at fjshull@sei.cmu.edu.

SAMUEL WEBER is a senior research staff member at Carnegie Mellon University's Software Engineering Institute, where he's a member of the Science of Cyber-security group. Contact him at samweber@cert.org.

See www.computer.org/software-multimedia for multimedia content related to this article.