Dr. Dobbs Jolt Award Finalist 2014

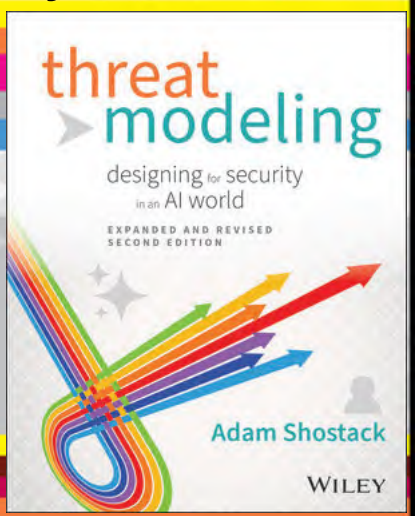
Adam Shostack
Microsoft's Threat Modeling Expert

Appendix D "Elevation of Privilege: The Cards" 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

APPENDIX

D

Elevation of Privilege: The Cards

This appendix discusses the threats in the *Elevation of Privilege* card game. It goes beyond the material included in the game, with the goal of making the game more helpful. Each bulleted item in the lists that follow includes the card, the threat, a brief discussion, and possibly a reference or comments about how to address it.

The aces are all of the form "You've invented a new type of attack." The intent of "new" is "not clearly covered in a card," rather than "never before seen." How to interpret "clearly covered" is up to the group that's playing. You might be liberal, and encourage use of the aces, especially by those who are not security experts. You might be harsh, and set a high bar for security experts. It depends on the tone of your gameplay. However you decide to interpret it, be sure to write down the threat and address it. There is no per-threat-type discussion of Aces.

For more on the motivation, design, and development of *Elevation of Privilege*, see my paper "Elevation of Privilege: Drawing Developers into Threat Modeling" (Shostack, 2012).

Spoofing

Many of the concepts here are discussed at length in Chapter 14 "Accounts and Identity."

2 of Spoofing. *An attacker could squat on the random port or socket that the server normally uses*. Squatting is a term of art for a program that occupies the resource before your program starts. If you use a random port (registered with some portmapper), how can a client ensure that they're connecting to the right place? If you use a named object or a file in /tmp, the same sort of issues will apply. You can address this by using ACLs to ensure that the named object is restricted to your code, and that it is not transient (that is, it exists regardless of whether your code is running). You can also use an object in a private directory, rather than /tmp. If you use a port, you'll need to authenticate after connection, as other programs can start listening on that port. Unix systems have reserved ports on which only root can listen, but using that requires that your code runs as root or SUID root; or, if you break your code into a privileged listener and a larger unprivileged processor, the root component has to synchronize over some mechanism that then may be vulnerable to squatting.

3 of Spoofing. *An attacker could try one credential after another and there's nothing to slow them down* (*online or offline*). This refers to brute-force attacks. Online as a term of art here means all attacks for which some code you've written has a chance to intercede; offline means the attacker has a copy of the datastore on which your authentication code relies, and the attacker can use whatever attack tools they'd like. Resisting online attacks involves backoff and possibly lockout, while resisting offline attacks involves salts and iterated hashes (as discussed in Chapter 14.

4 of Spoofing. *An attacker can anonymously connect because we expect authentication to be done at a higher level*. This may be a reasonable assumption, but have you validated it?

5 of Spoofing. *An attacker can confuse a client because there are too many ways to identify a server*. This can refer to the human or software clients. If your systems have several names (for example, a WINS name, a DNS name, and a branding name), what ensures that the client always sees the same thing?

6 of Spoofing. *An attacker can spoof a server because identifiers aren't stored on the client and checked for consistency upon reconnection* (*that is, there's no key persistence*). This refers to "trust on first use," (TOFU) as performed by SSH and other protocols.

7 of Spoofing. *An attacker can connect to a server or peer over a link that isn't authenticated* (*and encrypted*). This can result in the server, peer, or client being spoofed.

8 of Spoofing. *An attacker could steal credentials stored on the server and reuse them* (*for example, a key is stored in a world-readable file*). Or the key could be stored so that only a few principals can read it, but it's still vulnerable to theft. You may want to use hardware that is designed to hold high-value keys, rather than store them on the file system.

9 of Spoofing. *An attacker who obtains a password can reuse it* (*use stronger authenticators*). This may have a complex interplay with your requirements, but it's always worth asking whether passwords can be removed from the system. (There's a discussion of those trade-offs in Chapter 14.)

10 of Spoofing. *An attacker can choose to use weaker or no authentication*. Many systems have negotiated authentication, sometimes including no authentication. Consider the options that you want or need to make available.

Jack of Spoofing. *An attacker could steal credentials stored on the client and reuse them*. There's a wide variety of ways that credentials are stored on clients, and you cannot directly prevent any of them. These mechanisms include primarily cookies and passwords. Consider additional authentication mechanisms such as IP addresses or device fingerprinting, and be careful to not break the person's mental model. Also remember that many people delete cookies.

Queen of Spoofing. *An attacker could go after the way credentials are updated or recovered* (*account recovery doesn't require disclosing the old password*). Backup authentication is hard, so don't miss the discussion of threats in Chapter 14.

King of Spoofing. *Your system ships with a default admin password and doesn't force a change*. If your systems all have the same default admin password, it will end up on the web and available to unskilled attackers. If you use an algorithm to determine it, such as the media access control (MAC) address, or some hash of the address, that too will likely become known.

Tampering

Many of the issues brought up by tampering threats are addressed with crypto, covered in depth in Chapter 16 "Threats to Cryptosystems."

2 of Tampering. There is no 2 of Tampering card, as we were unable to find a tampering threat we thought would be common enough to warrant a card. Suggestions to the author are welcome, care of the publisher, or via various social media. (Naming platforms is attractive, and at odds with my aspiration towards having written a classic text.)

3 of Tampering. *An attacker can take advantage of your custom key exchange or integrity control that you built instead of using standard crypto*. Creating your own cryptosystem can be fun, but putting it into production is foolhardy.

4 of Tampering. *Your code makes access control decisions all over the place, rather than with a security kernel*. A security kernel (sometimes called a reference monitor) is a single place where all access control decisions can be made. The advantage

of creating one is consistency, and the forcing function of considering what the API should be, which means you need to consider the appropriate parameters in each authentication.

5 of Tampering. *An attacker can replay data without detection because your code doesn't provide time stamps or sequence numbers*. Generally, time stamps are harder to use well, because they require synchronized clocks. Also, time stamps may have a larger attack surface than sequence numbers used only for your purposes. (Additionally, someone might attack your clock for unrelated reasons, so again, time stamps are riskier.)

6 of Tampering. *An attacker can write to a datastore on which your code relies*. If you like it, then you should have put an ACL on it.

7 of Tampering. *An attacker can bypass permissions because you don't make names canonical before checking access permissions*. If your code checks permissions on ./rules, and attackers can make ./rules a link, then they can add whatever permissions they would like. Expand that to a full and canonical path.

8 of Tampering. *An attacker can manipulate data because there's no integrity protection for data on the network*. Generally, the ways to fix this will be SSL, SSH, or IPsec.

9 of Tampering. *An attacker can provide or control state information*. If your system relies on something that an attacker can change, such as a URL parameter of authenticated=true or username=admin, then a redesign is probably in order.

10 of Tampering. *An attacker can alter information in a datastore because it has weak ACLs or includes a group that is equivalent to everyone* (*"all Live ID holders"*). These two examples (of weak ACLs and everyone groups) are not exactly the same things, but we wanted to get them both in. The fix to either involves changing the permissions.

Jack of Tampering. *An attacker can write to some resource because permissions are granted to the world or there are no ACLs*. This is a slightly more broad version of number 10.

Queen of Tampering. *An attacker can change parameters over a trust boundary and after validation* (*for example, important parameters in a hidden field in HTML, or passing a pointer to critical memory*). Generally, you address these with pass-by-value, rather than pass-by-reference. You validate and use the values you're passed.

King of Tampering. *An attacker can load code inside your process via an extension point*. Extension points are great. It's very hard to create systems that load someone else's code in a library without exposing yourself to security (and reliability) risks.

Repudiation

Many of these threats are threats to logging, as logging is an essential part of non-repudiation. Repudiation threats are often an interesting foil for requirements, but they are covered less well by *Elevation of Privilege*.

2 of Repudiation. *An attacker can pass data through the log to attack a log reader, and there's no documentation regarding what sorts of validation are done.* Attackers can be distinguished by what data elements they can insert. Any web user can insert a URL into your HTTP logs by requesting it. The time stamp field is under the control of (a possibly subverted) web server. Your logs should distinguish who can write what.

3 of Repudiation. *A low privilege attacker can read interesting security information in the logs.* You should ensure that interesting security information is stored in logs that are protected.

4 of Repudiation. *An attacker can alter digital signatures because the digital signature system you're implementing is weak, or uses MACs where it should use a signature.* This attack is less about logs, more about the use of crypto to either authenticate or provide non-repudiation. If you're using MACs (message authentication codes), those are generally based on symmetric crypto, and as such provide only integrity, not non-repudiation with respect to the other end of a connection.

5 of Repudiation. *An attacker can alter log messages on a network because they lack strong integrity controls.* Over a network, where the threat includes those from untrusted entities with network access, a MAC can provide integrity to support non-repudiation. This threat assumes both ends are trustworthy, unlike the 4 of Repudiation.

6 of Repudiation. *An attacker can create a log entry without a time stamp* (*or no log entry is time stamped*). If you trust the other end of a connection to provide time stamps, what happens when they don't?

7 of Repudiation. *An attacker can make the logs wrap around and lose data.* A challenge with logs is what to do when you have a lot of them. You can either lose data or availability (by shutting down when you can't log). You should make a decision based on which is appropriate for your system.

8 of Repudiation. *An attacker can make a log lose or confuse security information.* For example, if you have a system that compresses logs ("previous message repeated a gajillion times") does that work only on identical messages?

9 of Repudiation. *An attacker can use a shared key to authenticate as different principals, confusing the information in the logs.* This relates to the 4 of Repudiation, and showcases the value of asymmetric cryptography versus shared keys.

10 of Repudiation. *An attacker can get arbitrary data into logs from unauthenticated* (*or weakly authenticated*) *outsiders without validation.* If you have a centralized logging point, what does it record about where data comes from, and how does it authenticate those systems?

Jack of Repudiation. *An attacker can edit logs, and there's no way to tell* (*perhaps because there's no heartbeat option for the logging system*). You can heartbeat so that there's some indication logs were working, and there are more robust systems that use cryptography (often in the form of hash chains or trees).

Queen of Repudiation. *An attacker can say, "I didn't do that," and you would have no way to prove them wrong.* This is a business requirement threat, and your logs must capture the sorts of things that your customers might deny having done.

King of Repudiation. *The system has no logs*. (It's tempting to say that this one is self-explanatory, but *Elevation of Privilege* is designed to draw developers into threat modeling.) Logs can be helpful not only in debugging and operations, but also in attack detection or reconstruction, and in helping to settle disputes. Your logs should be designed to address each of those scenarios.

Information Disclosure

Many information disclosure threats are best addressed with cryptography.

2 of Information Disclosure. *An attacker can brute-force file encryption because no defense is in place* (example defense: password stretching). Password stretching refers to taking a password and iterating over it thousands of times to make a better cryptographic key, which is then used to encrypt the document (rather than using the password directly).

3 of Information Disclosure. *An attacker can see error messages with security-sensitive content*. For example, your web error page says "Cannot connect to database with password foobar1." The right pattern is a unique error code and perhaps pointing to the relevant logs. This can also relate to the 3 of Repudiation (a low-privilege attacker can read interesting security information in the logs).

4 of Information Disclosure. *An attacker can read content because messages* (*for example, an e-mail or HTTP cookie*) *aren't encrypted even if the channel is encrypted*. The distinction between channel and message encryption is important. The channel is something like an SMTP connection, and even if that's encrypted, data is in the clear at endpoints.

5 of Information Disclosure. *An attacker might be able to read a document or data because it's encrypted with a nonstandard algorithm*. Use standard cryptographic algorithms.

6 of Information Disclosure. *An attacker can read data because it's hidden or occluded* (*for undo or change tracking*) *and the user might forget that it's there*. Change tracking is a lovely feature, and modern Microsoft products tend to have a "prepare for sharing" sort of function. If your data formats have similar issues, you'll want similar functionality.

7 of Information Disclosure. *An attacker can act as a "man in the middle" because you don't authenticate endpoints of a network connection*. Failures to authenticate lead to failures of confidentiality. You'll generally need both.

8 of Information Disclosure. *An attacker can access information through a search indexer, logger, or other such mechanism*. This can refer to a local search indexer, such as the Mac OS Spotlight; or a web indexer, such as Google.

9 of Information Disclosure. *An attacker can read sensitive information in a file with bad ACLs*. Bad ACLs! No biscuit! This really should have read "weak," rather than "bad," and the fix is more restrictive ACLs or permissions.

10 of Information Disclosure. *An attacker can read information in files with no ACLs*. Therefore, add some.

Jack of Information Disclosure. *An attacker can discover the fixed key being used to encrypt*. You likely don't use the same physical key to unlock every door you're authorized to open, so why use the same cryptographic key?

Queen of Information Disclosure. *An attacker can read the entire channel because the channel* (*for example, HTTP or SMTP*) *isn't encrypted*. As more and more data passes over untrustworthy networks, the need for encryption will continue to increase.

King of Information Disclosure. *An attacker can read network information because there's no cryptography used*.

Denial of Service

Threats 3–10 are constructed from three properties, shown in parentheses after the text description:

- **Is the threat to a client or a server?** Threats to servers likely affect more people.
- **Is the attacker authenticated or anonymous?** Threats in which an attacker needs credentials have a smaller pool of attackers (or require a preliminary step of acquiring credentials), and it may be possible to retaliate in some way, acting as a deterrent.
- **Does the impact go away when the attacker does (temporary versus persistent)?** Persistent issues that require manual intervention or destroy data are worse than threats that will clear up when the attacker leaves.

There is no discussion of these threats per card, but the cards are listed for reference or use in checking aces.

2 of Denial of Service. *An attacker can make your authentication system unusable or unavailable*. This refers to authentication systems that use either backoff or account lockout to prevent brute-force attacks. Fixing the issues raised by the 3 of Tampering can lead you here.

3 of Denial of Service. *An attacker can make a client unavailable or unusable but the problem goes away when the attacker stops* (*client, authenticated, temporary*).

4 of Denial of Service. *An attacker can make a server unavailable or unusable but the problem goes away when the attacker stops* (*server, authenticated, temporary*).

5 of Denial of Service. *An attacker can make a client unavailable or unusable without ever authenticating, but the problem goes away when the attacker stops* (*client, anonymous, temporary*).

6 of Denial of Service. *An attacker can make a server unavailable or unusable without ever authenticating, but the problem goes away when the attacker stops* (*server, anonymous, temporary*).

7 of Denial of Service. *An attacker can make a client unavailable or unusable and the problem persists after the attacker goes away* (*client, authenticated, persistent*).

8 of Denial of Service. *An attacker can make a server unavailable or unusable and the problem persists after the attacker goes away* (*server, authenticated, persistent*).

9 of Denial of Service. *An attacker can make a client unavailable or unusable without ever authenticating, and the problem persists after the attacker goes away* (*client, anonymous, persistent*).

10 of Denial of Service. *An attacker can make a server unavailable or unusable without ever authenticating, and the problem persists after the attacker goes away* (*server, anonymous, persistent*).

Jack of Denial of Service. *An attacker can cause the logging subsystem to stop working.* An attacker who can cause your logging to stop can execute attacks that are then harder to understand and possibly harder to remediate.

Queen of Denial of Service. *An attacker can amplify a denial-of-service attack through this component with amplification on the order of 10:1.* Amplification refers to the defender's resource consumption versus the attacker's. An attacker who just sends you a lot of data is consuming bandwidth at a ratio of 1:1. An attacker who sends a DNS request for a public key is sending dozens of bytes and receiving hundreds, so there's an amplification of 10:1 or so.

King of Denial of Service. *An attacker can amplify a denial-of-service attack through this component with amplification on the order of 100:1.* As per the Queen, but tenfold worse.

Elevation of Privilege (EoP)

2–4 of Elevation of Privilege. There are no cards for the 2, 3, or 4 of Elevation of Privilege, as we were unable to find EoP threats we thought would be common enough to warrant cards. Suggestions are welcome.

5 of Elevation of Privilege. *An attacker can force data through different validation paths which give different results.* If you have different code performing similar validation, then it's hard for your other functions to know what will be checked. This is a great opportunity to refactor.

6 of Elevation of Privilege. *An attacker could take advantage of .NET permissions you ask for but don't use.* The .NET Framework is an example; since the game was created, frameworks with permissions have become quite trendy, appearing in many mobile and even desktop operating systems. Asking for permissions you don't need reduces the security value of these frameworks.

7 of Elevation of Privilege. *An attacker can provide a pointer across a trust boundary, rather than data that can be validated*. This is the pass-by-reference/pass-by-value issue yet again. If you're going to make trust decisions, you need to ensure that the resources on which you're acting are outside the control of a potential attacker. This issue often appears when pointers are used to make kernel/userland or interprocess communication faster.

8 of Elevation of Privilege. *An attacker can enter data that is checked while still under the attacker's control and used later on the other side of a trust boundary*. This is a more open variant of the 7, using any data, rather than a pointer.

9 of Elevation of Privilege. *There's no reasonable way for callers to figure out what validation of tainted data you perform before passing it to them*. This can be solved by API design, so that fields are marked as "trustworthy" or not, or by documentation, which people are unlikely to read.

10 of Elevation of Privilege. *There's no reasonable way for a caller to figure out what security assumptions you make*. Perhaps this card should read "what security requirements you impose on them," but it's late for that. This card is a variant of the 9, intended to frame the issue because it's so frequent that extra chances to find it are good.

Jack of Elevation of Privilege. *An attacker can reflect input back to a user, such as cross-site scripting*. This card combines the 9 and 10, throwing in a trust boundary and some jargon. Here, an attacker finds a way to make data that comes from them appear to come from you, taking advantage of trust in you. Performing input and output validation can help here.

Queen of Elevation of Privilege. *You include user-generated content within your page, possibly including the content of random URLs*. This should be read more broadly than simply web pages (although there's a lot of fun to be had there). If what you think came from Alice came from Bob, what are the security implications of sending that back to Alice?

King of Elevation of Privilege. *An attacker can inject a command that the system will run at a higher privilege level*. Command injection attacks include use of control characters, such as the apostrophe or semicolon. When these are inserted in the right way to dynamic code, they can lead to an attacker being able to run code of their own choosing. You need to transform your input into a canonical form. Loop until the input doesn't transform anymore, then check it.